Note de cours pour exposé sur PVector
Voir aussi Stella Baruk, pp. 1274-1287 et Georges Alain, pp. 189-198

1) Définition générale

Un vecteur est à la fois une grandeur, une donnée d'orientation et de sens comme peut l'être une direction pour un trajet donné. Mémoriser un vecteur revient à se rappeler de la direction vers laquelle se trouve un autre point par rapport à un point d'origine, et quelle serait la distance du trajet menant à ce point. Puisqu'on peut parcourir un chemin de deux façons, le vecteur a toujours un sens, sauf pour le vecteur nul. Le vecteur n'étant pas situé selon des mesures absolues, on dit qu'il s'agit d'une pseudo-figure, car il n'a pas la fixité des autres configurations.

Wikipedia (http://fr.wikipedia.org/wiki/Vecteur)
Un vecteur est représenté par un segment orienté (une flèche) ayant pour extrémités un point de départ et un point d'arrivée. L'emplacement dans le plan ou l'espace n'a pas d'importance, deux déplacements de deux points d'origine distincts peuvent correspondre au même vecteur, seuls comptent sa longueur, sa direction et son sens. Il est donc possible de le faire glisser librement dans le plan, parallèlement à lui-même.
Une définition formelle utilise au préalable la notion de bipoint. Il est défini comme un couple de points. L'ordre a une importance : le premier point est appelé origine.

Remarque sur l'utilité des vecteurs en image de synthèse :
Certains ont recours à l'utilisation de vecteurs dans des projets où l'on souhaite reproduire des phénomènes physiques. Il simplifie le processus de simulation de forces et de mouvement (comme la vitesse ou l'accélération) et les calculs pouvant être effectués sur ces diverses propriétés. Même si ce n'est pas sont utilité première, il permet aussi de garder en mémoire des positions 2D ou 3D. Un seul objet suffit, alors que la mémorisation d'emplacement peut nécessiter plusieurs variables distinctes (posX, posY, posZ). Parmi les autres applications des vecteurs, notons l'orientation des figures (tournées vers un point donné) et leur alignement le long de chemins (tutoriel de Shiffman dans Nature of code).

2) Un vecteur dans Processing

```
PVector v1 = new PVector(100, 100, 0);
```

Notons d'abord quelques particularités dans l'affectation de valeurs à notre variable v1 (ce qui se trouve à droite du symbole =) :
- La classe PVector fonctionne avec des objets (POO ou programmation orientée objet). Ainsi, il faut «construire» notre instance en invoquant le «new», en lui rappelant bien quel est le nom de la classe, soit «PVector» et en plaçant entre parenthèses les deux ou trois arguments requis

(si on travaille en 2D, le troisième argument vaut simplement 0).
- En mode continu (void setup() + void draw()), et advenant la déclaration de variables globales, il est préférable de construire l'objet et de lui affecter ses valeurs dans le setup() ou le draw(). Ainsi, on devrait plutôt privilégier une écriture comme suit :

```
// Déclaration de la variable globale de type PVector
PVector v1;
// Puis ailleurs, dans la fonction setup() ou, plus rarement, dans le draw() :
v1 = new PVector(100, 100, 0);
```

Le vecteur ci-dessus pointe à la fois vers la droite dans l'axe des X, vers le bas dans l'axe des Y (puisqu'Y croît vers le bas dans un sketch Processing), puis ni vers l'avant et ni vers l'arrière dans l'axe des Z.

Dans l'approche vectorielle, la longueur est celle du segment séparant les bipoints (point d'origine, point d'extrémité). On emploiera le terme magnitude dans ce cas (pour le mettre en lien avec les méthodes dans Processing). Le terme français serait «norme».
La magnitude (la norme) est toujours plus grande ou égale à 0.

```
// Voici une fonction (la commande se termine par des parenthèses) applicable sur un vecteur
// Celle-ci retourne un float
print(v1.mag()); // imprime 141.42136
```

On peut attribuer une valeur spécifique à la magnitude. v1.setMag(200); // Cette méthode ne fonctionne que dans la version Beta (2.0b et +) de Processing On peut également procéder en deux étapes : - D'abord convertir le vecteur en vecteur unitaire v1.normalize(); - Puis le multiplier par une valeur scalaire (un nombre, et pas un vecteur, ce qui est automatique si on a un seul argument de type float).
v1.mult(200); Cette valeur devient la nouvelle distance entre les bipoints (origine-extrémité). Ce n'est pas une position absolue, sauf pour un vecteur colinéaire avec l'axe des X, dont le sens est dirigé vers la droite. Dans ce cas alors, et puisque v1.x retourne 200.0, on peut dire que cet attribut équivaut à sa position en X. En d'autres mots, il y aura une infinité de vecteurs de magnitude 200, la position à l'extrémité pouvant correspondre à tous les points d'un cercle dont le rayon serait de 200 pixels.

Si la magnitude du vecteur est induite par une variable susceptible de croître ou de produire des valeurs que l'on contrôle partiellement, il est souvent utile de fixer une borne maximale.
v1.limit(150);
Avec la méthode .limit(), la magnitude ne pourra pas dépasser la valeur prescrite. Notons que cette commande n'est pas définitive et elle doit être appelée chaque fois que la magnitude du vecteur a été modifiée.

Note : un vecteur unitaire (ou normé, ou «normalize») est un vecteur dont la magnitude équivaut à 1. Cette notion sera utile quand on veut simplement connaître la direction d'un trajet, pour lui appliquer ensuite une distance arbitraire plus grande ou plus petite à 1.

v1.normalize();
print(v1.mag()); // imprimera toujours une valeur approchant 1.0 (ici, 0.99999994)

Lire les «composants» x , y, ou z du vecteur revient à connaître l'emplacement vers lequel nous mènerait le trajet SI le point d'origine de ce dernier était situé à 0, 0, 0. Comme on comprend mieux le vecteur en le symbolisant par une flèche, lire les attributs x, y, et z d'un vecteur équivaut en quelque sorte à cibler le point placé à sa pointe. C'est seulement lorsque la base de cette flèche coïncide avec la coordonnée d'origine du plan cartésien que les attributs renvoient à une position dans ce plan.
Comme, par défaut, les coordonnées 0,0,0 d'un sketch sont situées dans le coin supérieur gauche, le positionnement d'un objet à l'aide des paramètres v1.x et v1.y d'un vecteur est relatif à cette origine. Si on veut qu'il soit relatif à un autre lieu, le centre du sketch par exemple, il faudra préalablement appliquer des transformateurs géométriques tels que translate(width/2, height/2, 0).

// En raison des changements apportés à la magnitude de v1, voici les nouvelles valeurs du point d'arrivée
println(v1.x); // imprime 0.70710677
println(v1.y); // imprime 0.70710677
println(v1.z); // imprime 0.0 On peut aussi consulter les valeurs de nos composants comme suit : v1.toString() // retourne [0.70710677, 0.70710677, 0.0] dans la console

// Si on veut réaffecter des valeurs à un objet PVector existant (une instance déjà déclarée) :
v1.set(100, 100, 0);

print(v1.x); // retourne 100.0
print(v1.y); // retourne 100.0
print(v1.z); // retourne 0.0

3) Opérations sur les vecteurs dans Processing

«Since vectors represent groupings of values, we cannot simply use traditional addition/multiplication/etc. Instead, we'll need to do some "vector" math, which is made easy by the methods inside the PVector class.» Tiré de la fiche PVector sur Processing.org

########## section en développement continuel ##############

3.1 Addition de vecteurs
// Méthode .add() // retourne un vecteur
Visuellement, cela équivaut à placer des flèches bout à bout et à tirer une nouvelle flèche de la première origine à la dernière extrémité. En additionnant deux vecteurs, on forme alors un triangle. En mathématiques, l'opération est connue sous le nom de Relation de Chasles.
Une autre approche consiste à laisser les deux flèches à la même origine et dessiner un parallélogramme. Une flèche tracée de l'origine au coin opposé du parallélogramme équivaut au résultat de l'addition.

3.2 Soustraction de vecteurs
// Méthode .sub() // retourne un vecteur
Visuellement, cela équivaut à renverser le parallélogramme de l'opération d'addition. L'axe de renversement est celui du premier terme de la soustraction. Ainsi, dans v1 - v2, il faut imaginer le parallélogramme se trouvant de l'autre côté de v1.
Notons enfin que le vecteur résultant de la soustraction est «identique» à celui qui séparerait les deux pointes des flèches.

NOTE IMPORTANTE : Les méthodes add(), sub(), mult() et div() peuvent être employées de deux façons : suivant des méthodes dites «static» ou «non-static». Dans le premier cas, on prend soin de reléguer le résultat de l'opération à un troisième vecteur, laissant ainsi intacts les deux vecteurs qui ont été utilisés. PVector v3 = PVector.add(v1, v2); Dans le second cas (non-static), on doit noter que l'un des deux vecteurs se trouvera modifié à la suite de l'opération : v1.add(v2);

3.3 Multiplication, soit de deux vecteurs, ou plus simplement d'un vecteur avec une grandeur (ce qui change sa magnitude)
// Méthode .mult() // retourne un vecteur

3.4 Division
// Méthode .div() avec soit entre deux vecteurs, soit avec une grandeur scalaire

3.5 Opérations de comparaison entre deux vecteurs
3.5.1 Distance entre deux vecteurs
// Méthode .dist() // retourne un float
```
PVector v1 = new PVector(100, 100, 0);
PVector v2 = new PVector(0, 200, 0);
float d = PVector.dist(v1, v2); // retourne 141.42136
// peut aussi s'écrire
float d = v1.dist(v2);
```

3.5.2 Angle entre deux vecteurs

```
// Méthode .angle() // retourne un float
PVector v1 = new PVector(100, 100, 0);
PVector v2 = new PVector(-50, 50, 0);
float a = PVector.angleBetween(v1, v2); // angle en radians
println(degrees(a)); // retourne 90.0 degrés
```

La valeur maximale pouvant être retournée est de PI ou 180 degrés, soit deux vecteurs en direction opposée.

Des multiplicateurs particuliers:

3.6 Produit scalaire (dot product)

Méthode .dot(); // retourne un float

3.7 Produit vectoriel (cross product)

Méthode .cross(); // retourne un vecteur Cet opérateur est utile pour établir les tangentes d'un déplacement circulaire.

4) Exemple d'applications

4.1 Quelle opération permet de faire pointer un vecteur dans sa direction opposée?

```
PVector v1 = new PVector (100,0,0);
v1.mult(-1);
println(v1.toString()); // retourne [ -100.0, -0.0, -0.0 ]
```

Sinon, quelle opération permet d'annuler l'effet d'une force? v1.mult(0);

4.2 On souhaite définir un vecteur qui soit toujours orienté depuis un point donné (le centre du sketch, par exemple) vers la position du curseur, voire au=delà?

Q. Quel est le vecteur pouvant relier les pointes de deux autres vecteurs? On trouvera ce vecteur avec l'opérateur de soustraction.

```
PVector centreSketch = new PVector(width/2, height/2, 0);
PVector posCurseur = new PVector(mouseX, mouseY, 0);
PVector v3 = PVector.sub(posCurseur, centreSketch);
// Et pour le dessiner
translate(centreSketch.x, centreSketch.y);
line(0, 0, v3.x, v3.y);
// On peut aussi essayer ceci, et voir que l'orientation de la ligne est toujours maintenue
translate(centreSketch.x, centreSketch.y);
translate(v3.x, v3.y);
```

```
v3.setMag(100);
line(0, 0, v3.x, v3.y);
```

4.3 On cherche à trouver le point mitoyen entre plusieurs vecteurs.
Voici un exemple de fonction, exploitée après la création d'un tableau contenant au moins un vecteur :

```
PVector moyenneVecteurs () {
        PVector moyenne = new PVector();
        for (int i =0; i < listeVecteurs.length; i++) {
                moyenne.add(listeVecteurs[i]);
        }
        return PVector.div(moyenne, listeVecteurs.length);
}
```