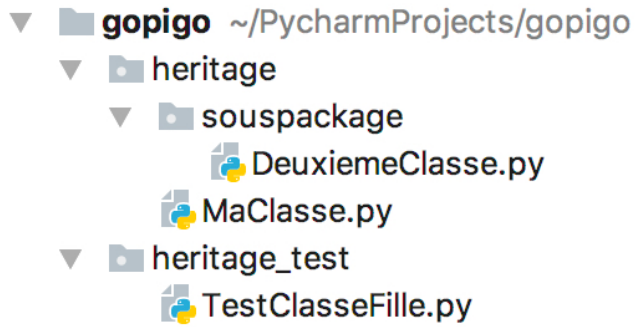
Cours 2 : Géométrie dans l'espace
Introduction à OpenGL

Vincent Guigue
UPMC - LIP6

Organisation du code en package

- Comme en Java : répertoire et sous-répertoire



- Accéder au code d'un fichier depuis un autre fichier :

```
1 from heritage.MaClasse import *
```

  - Accès aux classes & méthodes définies dans le fichier...
  - ... Et exécution des scripts

- L'astuce pour éviter l'exécution automatique des scripts

```
1 import ...
2
3 # blocage de l'exécution dans l'import
4 if __name__ == '__main__':
5         a = 2
6         ...
```

  - Possibilité de définir un fichier `__init__.py` dans le répertoire pour spécifier ce qui doit être importé ou pas

Organisation d'une classe

- Déclaration des attributs & protection (public / protected / private)

```
1  import numpy as np
2
3  class A:
4      def __init__(self):
5          self.__att_prive = 12
6          self._monde = np.array([[i+j for i in range(10)] \
7              for j in range (10)])
8          self.attrib = 18
9
10     def getSommeLigne1(self):
11         return self._monde[0].sum()
12
13     def print_natt(self):
14         print(self._natt) # aucun pb tant que
15                           # la méthode n'est pas appelée
16
17 if __name__ == '__main__':
18     une_instance = A()
19     print(une_instance.getSommeLigne1())
20     print(une_instance._monde)
21     print(une_instance.__att_prive)
```

Organisation d'une classe (2)

Philosophie générale

En python, tout est permis...

C'est à vous de faire attention

```python
class Cup:
    def __init__(self, color):
        self._color = color     # protected variable
        self.__content = None   # private variable

    def fill(self, beverage):
        self.__content = beverage

    def empty(self):
        self.__content = None
```

```python
redCup = Cup("red")
redCup._Cup__content = "tea"
```

⇒ A proscrire **absolument** !

Organisation d'une classe (3)

- Invocation des méthodes

```
une_instance.getSommeLigne1() # syntaxe JAVA
# OU
A.getSommeLigne1(une_instance) # syntaxe bizarre
                               # (qui explique la signature)
```

- Une approche extrêmement dynamique

```
    # code pré-existant dans la classe A
    class A:
    ...
    def print_natt(self):
        print(self._natt) # aucun pb tant que
                          # la méthode n'est pas appelée

    # Dans le main
    une_instance._natt = 9
    print(une_instance._natt)
    une_instance.print_natt()
```

Héritage

- Simple

```
1 from heritage.MaClasse import *
2
3 class B(A):
4     def __init__(self): # constructeur
5         A.__init__(self) # A FAIRE SOUS PEINE DE SURPRISE
6         self.unematrice = np.array([[i+j for i in range(3)] for j in
7
8     def print_mat(self):
9         print(self.unematrice)
```

- Multiple

```
1 class B(A, C):
2     def __init__(self): # constructeur
3         A.__init__(self) # A FAIRE SOUS PEINE DE SURPRISE
4         C.__init__(self) # A FAIRE SOUS PEINE DE SURPRISE
```

$\Rightarrow$ Conflits possibles

On cherchera les méthodes dans l'ordre de définition de la parenté

Polymorphisme

```
    # dans A
    def unefonction(self):
        print("coucou mere (A)")
    # dans B(A)
    def unefonction(self):
        print("coucou fille (B)")
```

Dans le main :

```
from heritage.MaClasse import *
from heritage.souspackage.DeuxiemeClasse import *

if __name__ == '__main__':
    une_instance = A()
    b = B()

    print(b.calcul())
    print(b._monde)

    li = [une_instance, b] # et si on ajoute ["toto"] ?
    for obj in li:
        obj.unefonction()
```

Conclusion & debug

DANGER

- Une syntaxe **très** permissive

⇒ C'est à vous de faire attention !

AVANTAGE

- Un énorme avantage : l'approche script, qui permet de debugger en ligne
  1. Test sur l'état de la mémoire en fin d'exécution
  2. Point d'arrêt dans le programme :

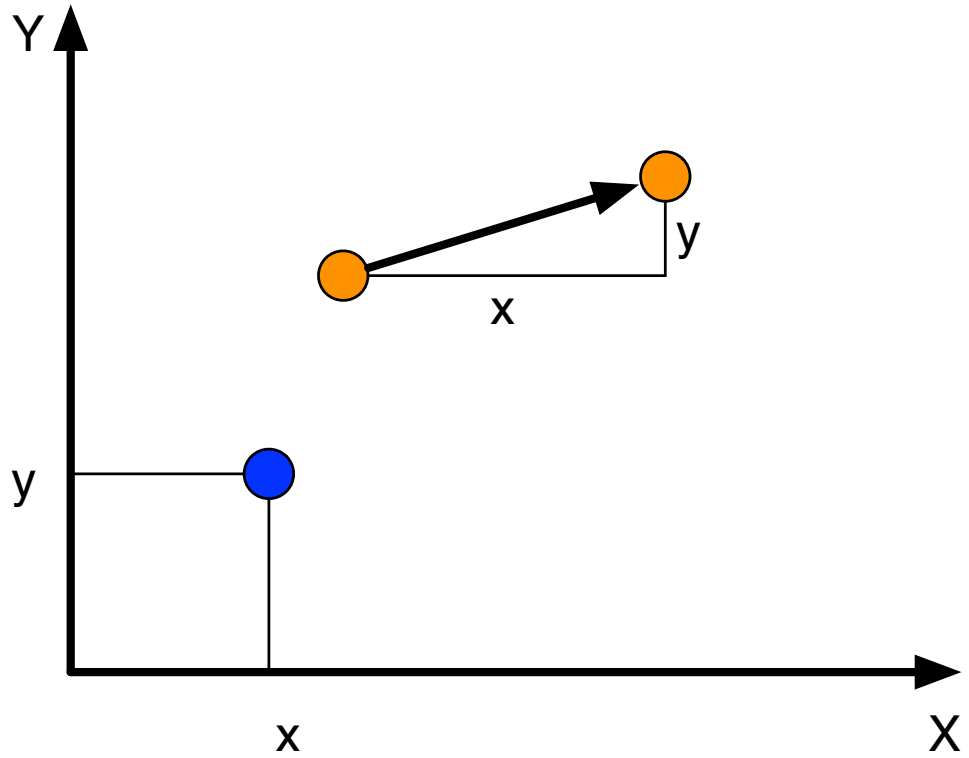
```
1 import pdb
2
3 ...
4 pdb.set_trace() # point d'arrêt
5 ...
```

Idée générale

- Définir entre vous
  - les interfaces de haut niveau vers les capteurs
  - les interfaces de haut niveau vers les actionneurs
- **Séparer le robot en blocs plus simples...**
- ... Mais conserver un modèle géométrique général
  - 1 robot = 1 rectangle qui se déplace
  - Pas de problématique de frottement, pas de modélisation complexe des actionneurs
  - Gérer les capteurs à part
- Besoin de quelques **outils** ...

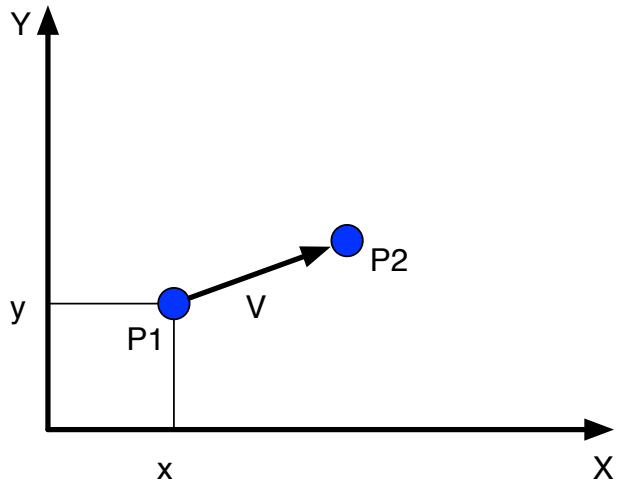
  que nous allons voir maintenant
  - Géométrie dans le plan (2D)
  - Modélisation 3D
  - Gestion des images

Repérage dans l'espace 2D



- A l'aide de la classe `Point`
  - Attributs `double` x et y
- La même classe nous permet de gérer les points et les vecteurs

Gestion des déplacements

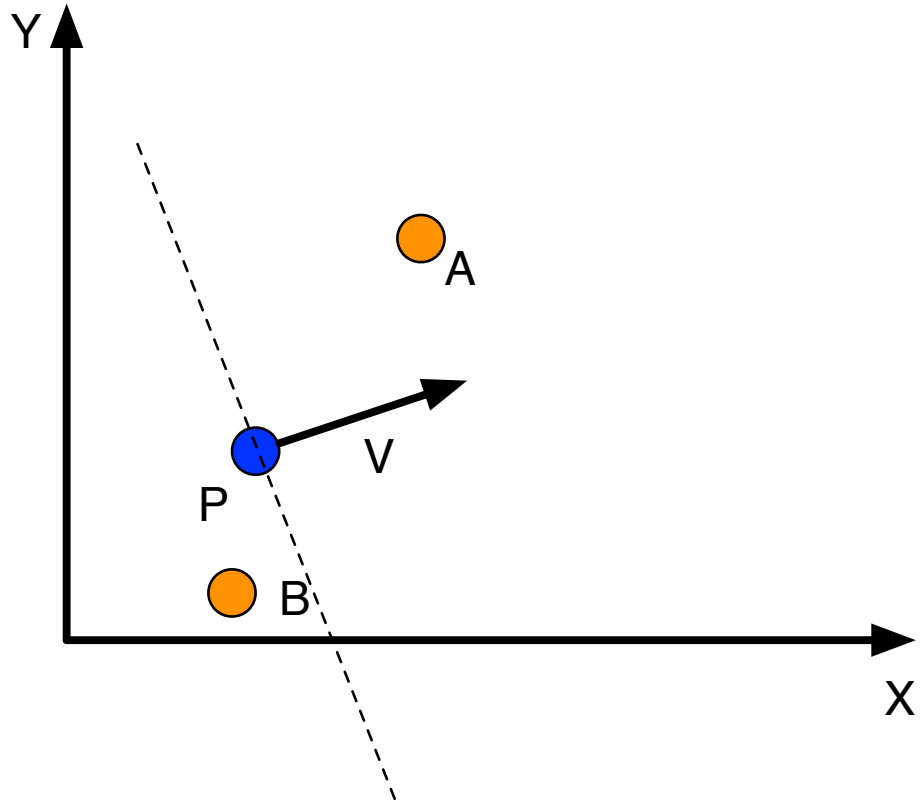


- Déplacements discrets (P : position, V : vitesse) :

$$P_2 = P_1 + V, \quad \begin{cases} P_2.x = P_1.x + V.x \\ P_2.y = P_1.y + V.y \end{cases}$$

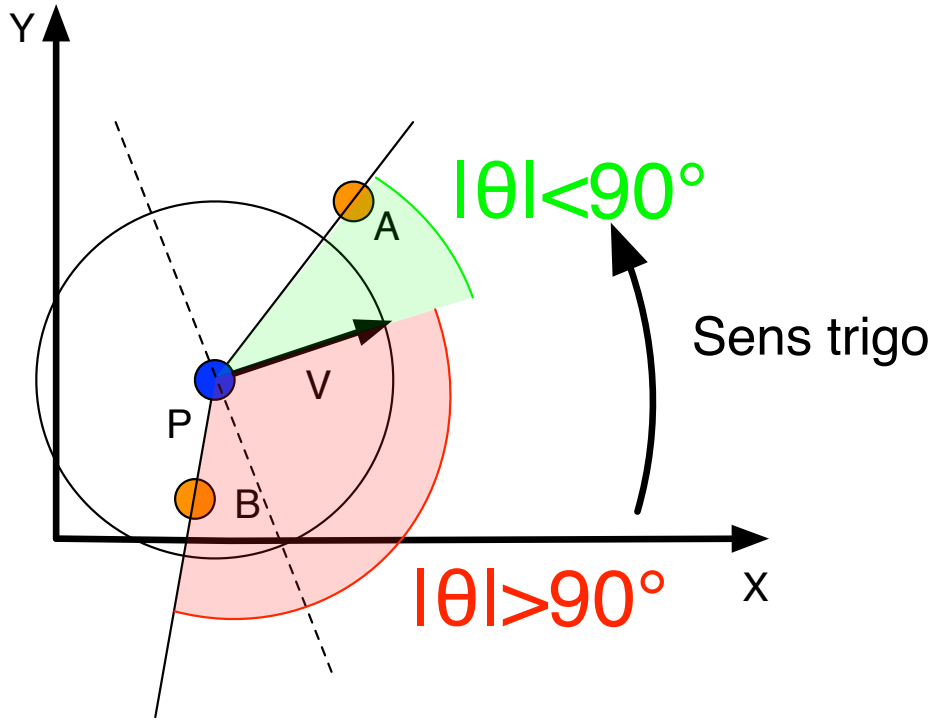
- En physique : $\vec{v} = \dot{\vec{x}} \approx \frac{\vec{x}_{t+1} - \vec{x}_t}{\delta_t}$, pour nous : $\delta_t = 1$ (unité arbitraire)
- En utilisant des vecteurs suffisamment petits : modélisation d'un déplacement continu

Se repérer dans l'espace



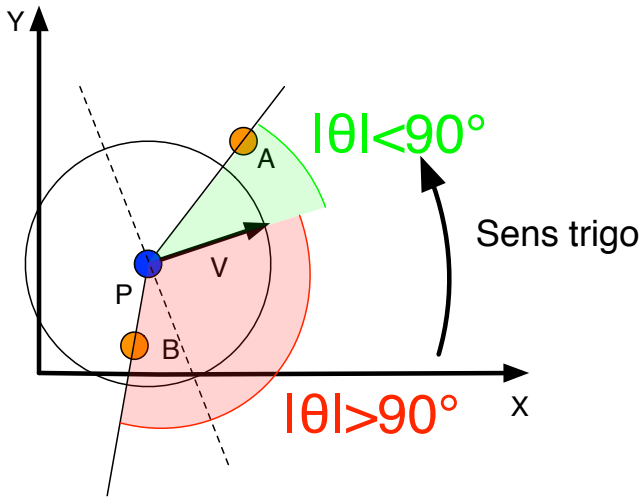
- Un objet est caractérisé par sa position P et sa vitesse V
- Qu'est ce qui est devant, qu'est ce qui est derrière l'objet ?
- Qu'est ce qui est à droite, qu'est ce qui est à gauche ?

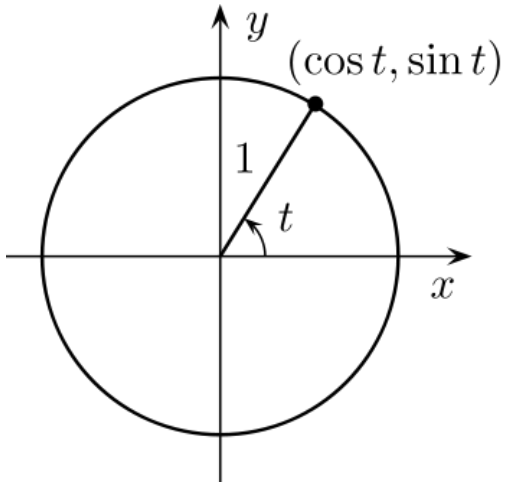
Solution complète : calcul d'angle



- Devant/Derrière : calculer les angles $\widehat{V, PA}$ et $\widehat{V, PB}$
- Gauche/Droite : calculer les angles $\widehat{V, PA}$ et $\widehat{V, PB}$ avec le signe !

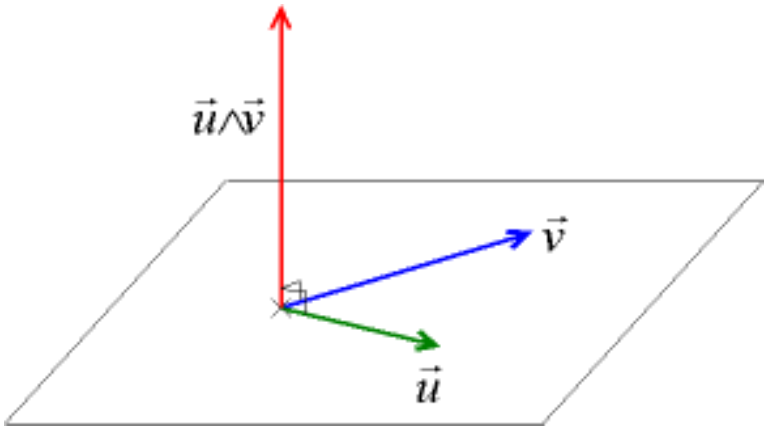
Détail du calcul d'angle

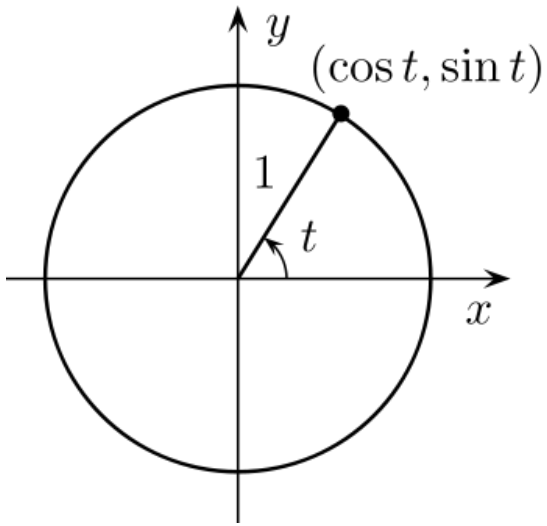




- Produit scalaire
  $U \cdot V = \langle U, V \rangle = \|U\|\|V\| \cos(\widehat{U, V}) = U_x V_x + U_y V_y$
- Corollaire :
$$\widehat{U, V} = \text{acos}\left(\frac{U \cdot V}{\|U\|\|V\|}\right)$$
- Attention : $\widehat{U, V} \in [0,\ \pi]$, pas de signe ici...
- Le signe de $U \cdot V$ permet de résoudre le pb devant/derrière

Détail du calcul d'angle (suite)





- Produit vectoriel : $\|U \wedge V\| = \|U\|\|V\| \sin(\widehat{U, V})$
- Corollaire :

$$\widehat{U, V} = \text{asin}\left(\frac{U \wedge V}{\|U\|\|V\|}\right) \qquad U \wedge V = \begin{bmatrix} u_2 v_3 - u_3 v_2 \\ u_3 v_1 - u_1 v_3 \\ u_1 v_2 - u_2 v_1 \end{bmatrix}$$

- L'étude de $u_1 v_2 - u_2 v_1$ permet de connaitre le signe de l'angle...

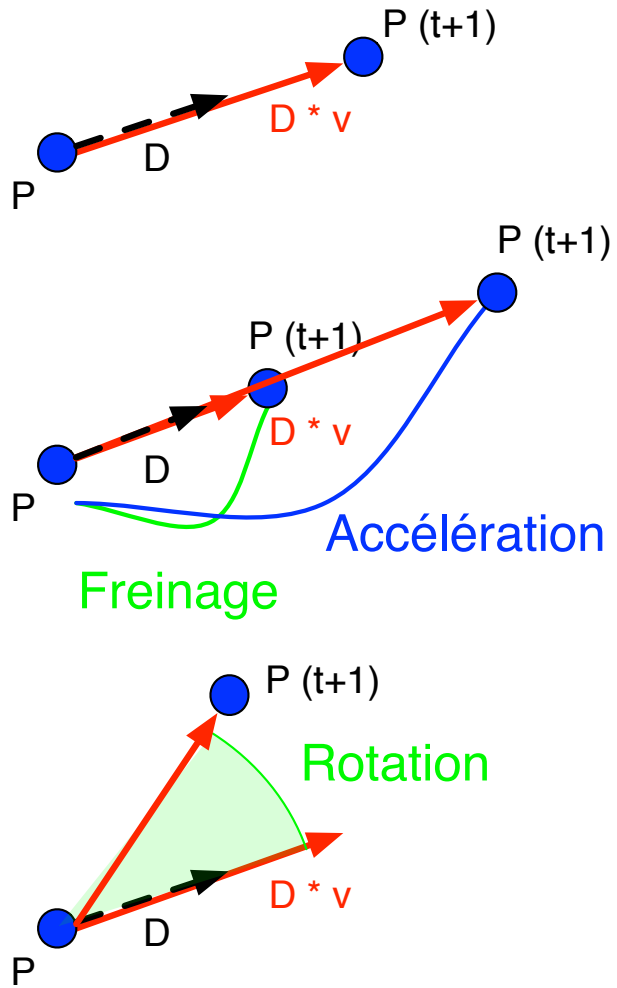
Pré-définition du robot

Le robot sera défini géométriquement par :

- Sa position : P
- Sa direction (vecteur unitaire) : D
  Conservation de la direction même à l'arrêt
- Sa vitesse (scalaire) : $v \in [0,\ vmax]$

La commande de la voiture se fera sur 2 axes :

- Accélération/Freinage : modification de v
- Commande de direction : modification de D



Rotation sur un vecteur

Définition de la rotation d'un vecteur :

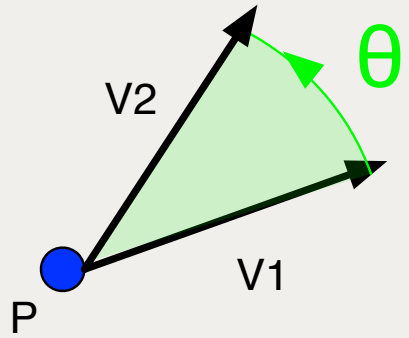
Soit un vecteur $V = \begin{bmatrix} v_x \\ v_y \end{bmatrix}$, la rotation d'angle θ est obtenu en utilisant la matrice de rotation :

$$R = \begin{bmatrix} \cos(\theta) & -\sin(\theta) \\ \sin(\theta) & \cos(\theta) \end{bmatrix}, \quad V' = RV$$

C'est à dire en utilisant la mise à jour :

- $v'_x = v_x \cos(\theta) - v_y \sin(\theta)$
- $v'_y = v_x \sin(\theta) + v_y \cos(\theta)$

ATTENTION à ne pas modifier v_x avant la seconde ligne



Rotation sur un rectangle

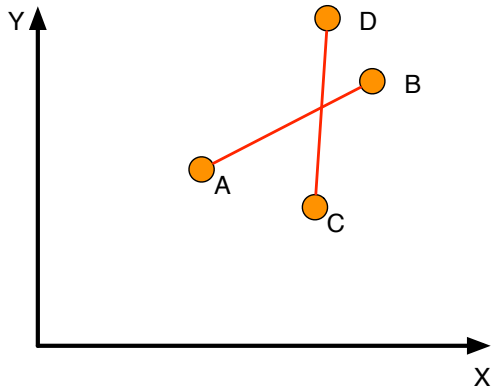
Comment appliquer une rotation de centre E sur un rectangle (A, B, C, D) ?

Rotation sur un rectangle

Comment appliquer une rotation de centre E sur un rectangle (A, B, C, D) ?

$\Rightarrow$ 4 rotations sur les vecteurs EA, EB, EC, ED

Approfondissement : collisions



(Problématique de base dans les cartes graphiques/moteur physique)

Comment détecter la collision de deux vecteurs ?

- Si C et D sont à gauche et à droite de AB
- ET que A et B sont à gauche et à droite de CD

Résultat :

S'ils sont de part et d'autre, l'un des produit vectoriel est positif, l'autre négatif...

$$(AB \wedge AC)(AB \wedge AD) < 0 \text{ ET } (CD \wedge CA)(CD \wedge CB) < 0$$

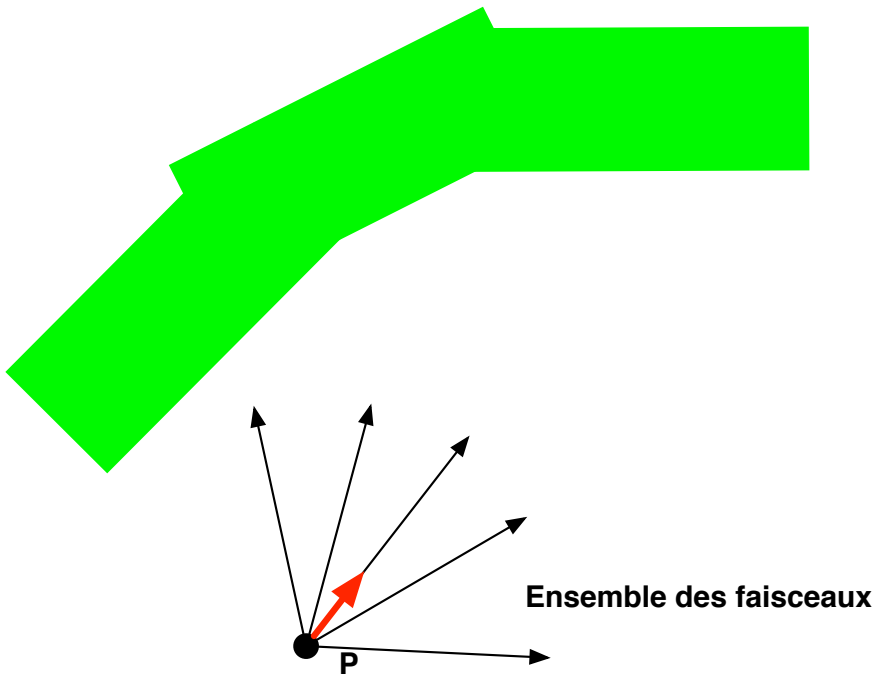
Résumé des méthodes de la classe Vecteur

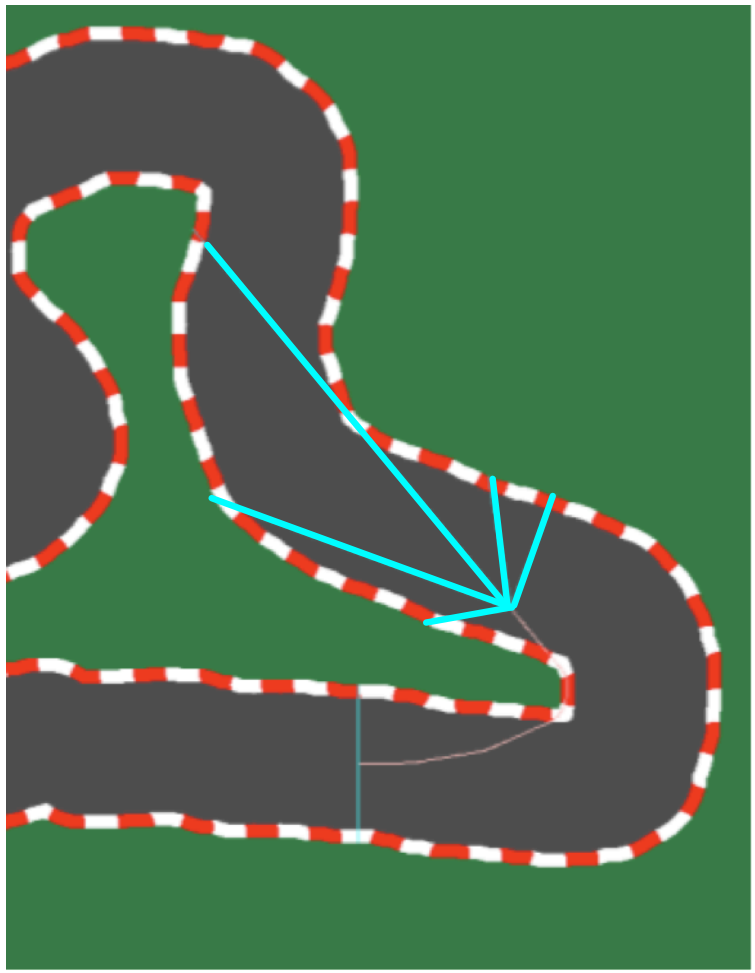
- Addition, soustraction
  - génération d'un nouveau vecteur
  - auto-opérateur
- produit scalaire
- produit vectoriel (composante en z)
- multiplication par un scalaire
- rotation
- calcul de la norme
- clonage
- test d'égalité (structurelle)

Radar : Fonctionnement

Principe

Le radar permet de se diriger à la manière d'un sous-marin, en envoyant des échos dans différentes directions.

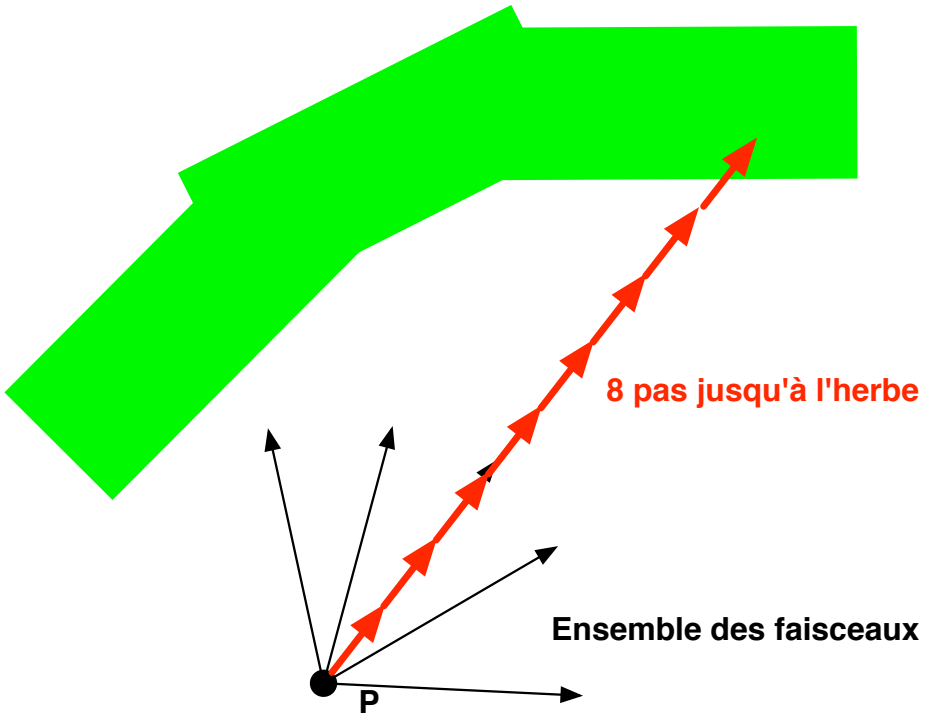




Radar : un peu plus de détails

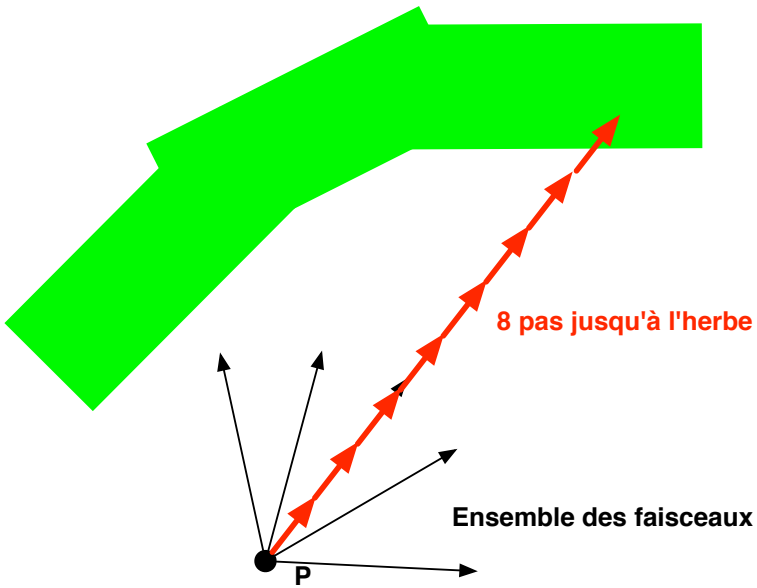
- Partir de la voiture
- Dans la direction D
- Avancer d'un pas EPS
- Recommencer tant qu'il n'y a pas d'obstacle
- Renvoyer le nombre de pas effectués

Appliquer une rotation sur D et recommencer



Radar : algorithme de base

Data: Voiture v, Circuit c, EPSILON
Result: cpt (score, plus c'est grand, plus c'est intéressant)
begin
    $P \longleftarrow v.position$;
    $D \longleftarrow v.direction$;
    $cpt \longleftarrow 0$;
    while *Cicuirt.getTerrain(P) != Herbe && P dans le circuit* **do**
        cpt++;
        $P \longleftarrow P + D * EPSILON$
    end
end



Radar : Cahier des charges

- Interface de définition d'un radar
  - Obtenir les scores de chaque branche
    NB : score le plus grand = le meilleur
  - Connaître la distance associée à chaque branche
  - Obtenir la meilleure branche
  - Récupérer les angles associés à chaque branche

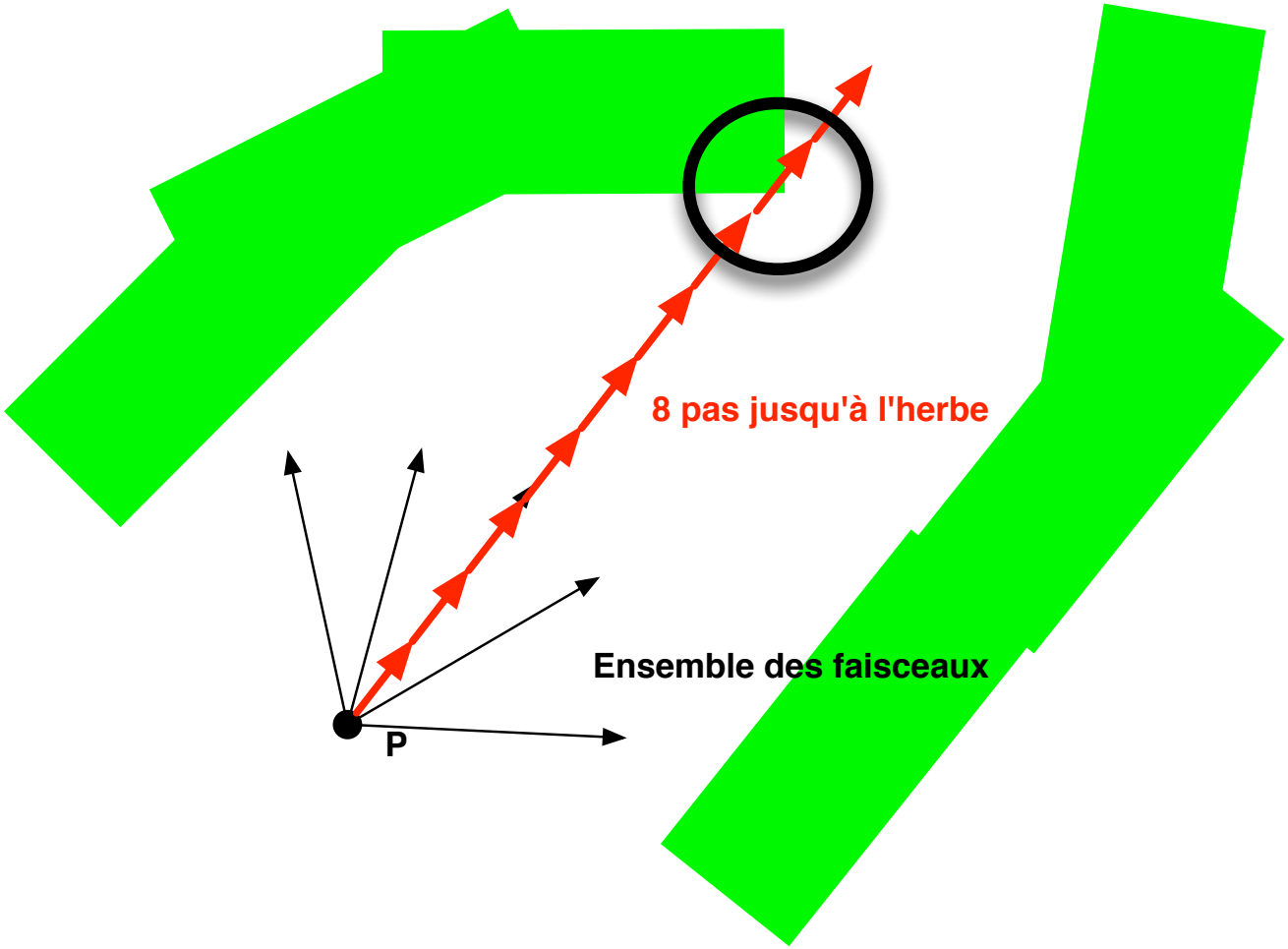
Radar : Piège d'implémentation

Attention aux copies de surfaces ! ! !

- Suivant les implémentations précédentes, on retourne parfois directement la position/direction de la voiture... Gare aux modifications !
- Même phénomène avec le point de départ du circuit dans la voiture

Radar : piège géométrique

Il ne faut pas prendre EPSILON trop petit : temps de calcul...
Il ne faut pas prendre EPSILON trop grand :



Le problème du bruit

- Dans le modèle
- Dans les données des capteurs

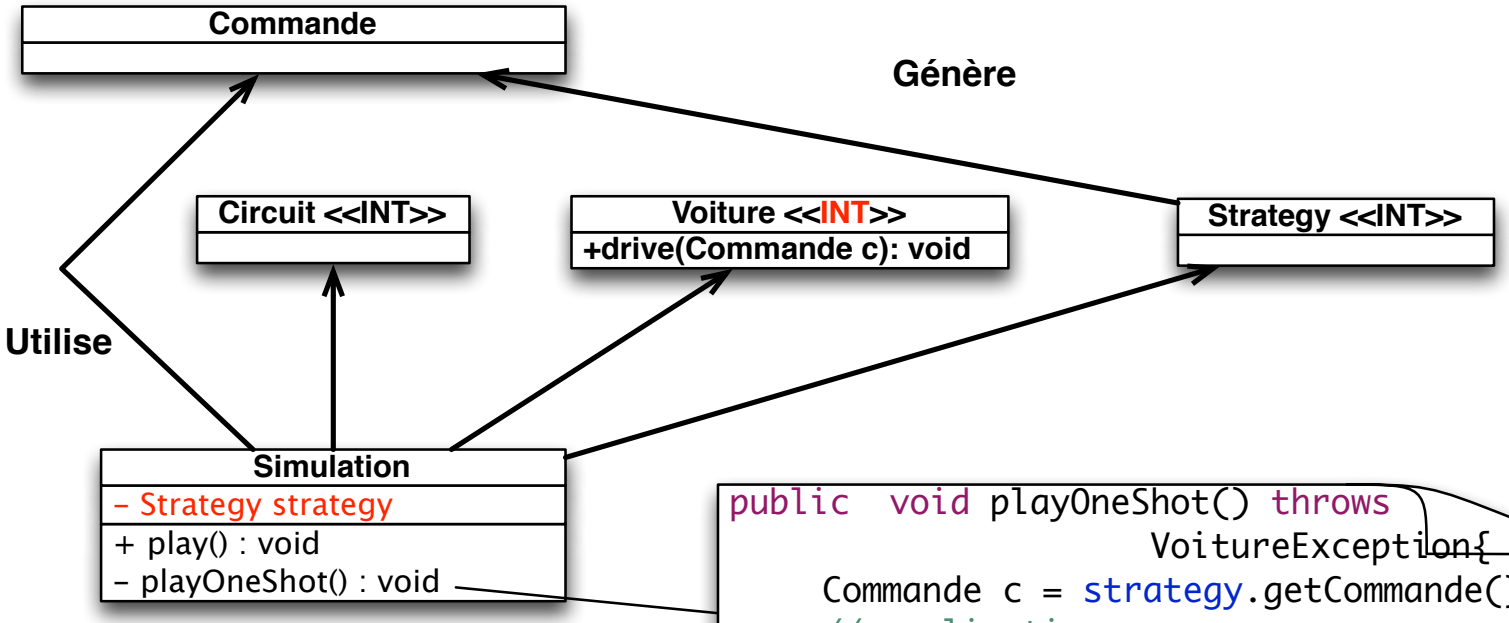
$\Rightarrow$ modélisation aléatoire

2 classes intéressantes

- `random`
  - bruit uniforme, tirage d'entier, tirage gaussien
- `numpy.random`
  - Gestion de matrice aléatoire
  - Tirage selon la plupart des lois usuelles

Architecture actuelle

- Informations stockées dans une image
- Création de l'image :
  - Soit dans le main
  - Soit dans la Simulation
- Modification de l'image :
  - Dans la simulation
- Sauvegarde de l'image :
  - Dans le main

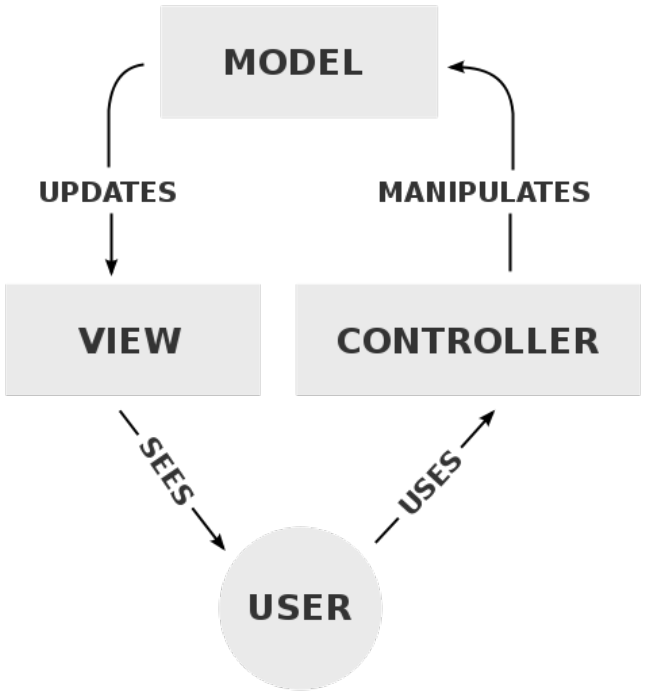



```
public  void playOneShot() throws
                        VoitureException{
    Commande c = strategy.getCommande();
    // application
    voiture.drive(c);
    // MAJ Etat
    state = updateState();
}
```

Vers un système indépendant

- On veut un système indépendant
  - Pouvoir le brancher ou le débrancher facilement, sans intervenir dans le code de la simulation
  - Pouvoir changer de système de visualisation
- Il existe un modèle standard (assez lourd) pour gérer cette situation : MVC Model View Controller

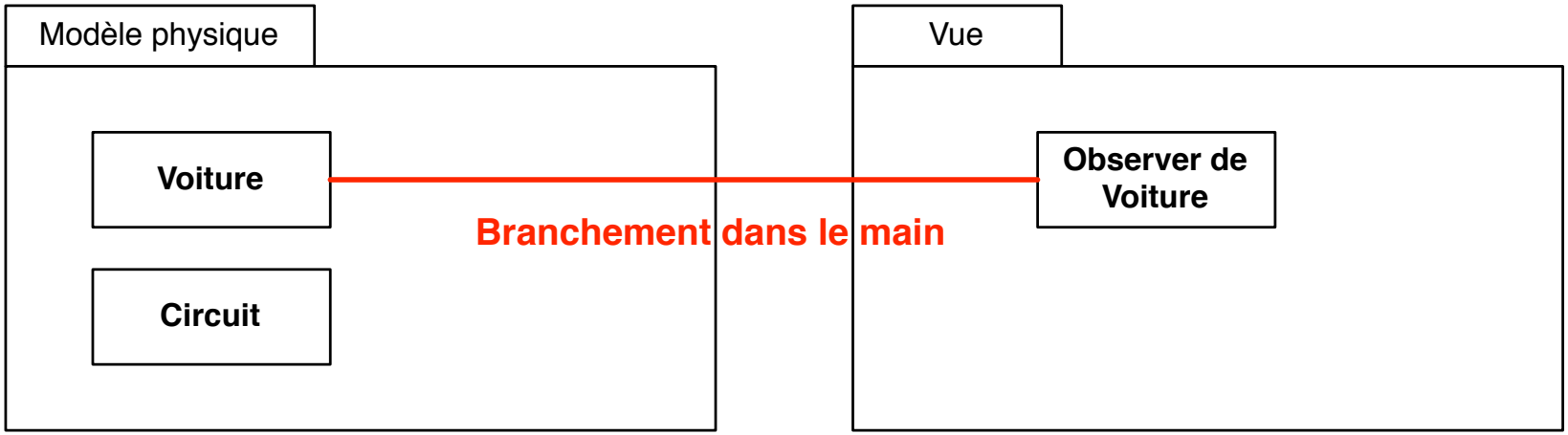
- Pour chaque élément (voiture, circuit,...), un objet vue est créé.
- Une vue générique est élément capable de gérer la vue d'un objet (cf slide suivant)
- Lorsque le modèle change il informe le Controller (évènement)
- Le Controller met à jour les informations d'affichage



credit : wikipedia

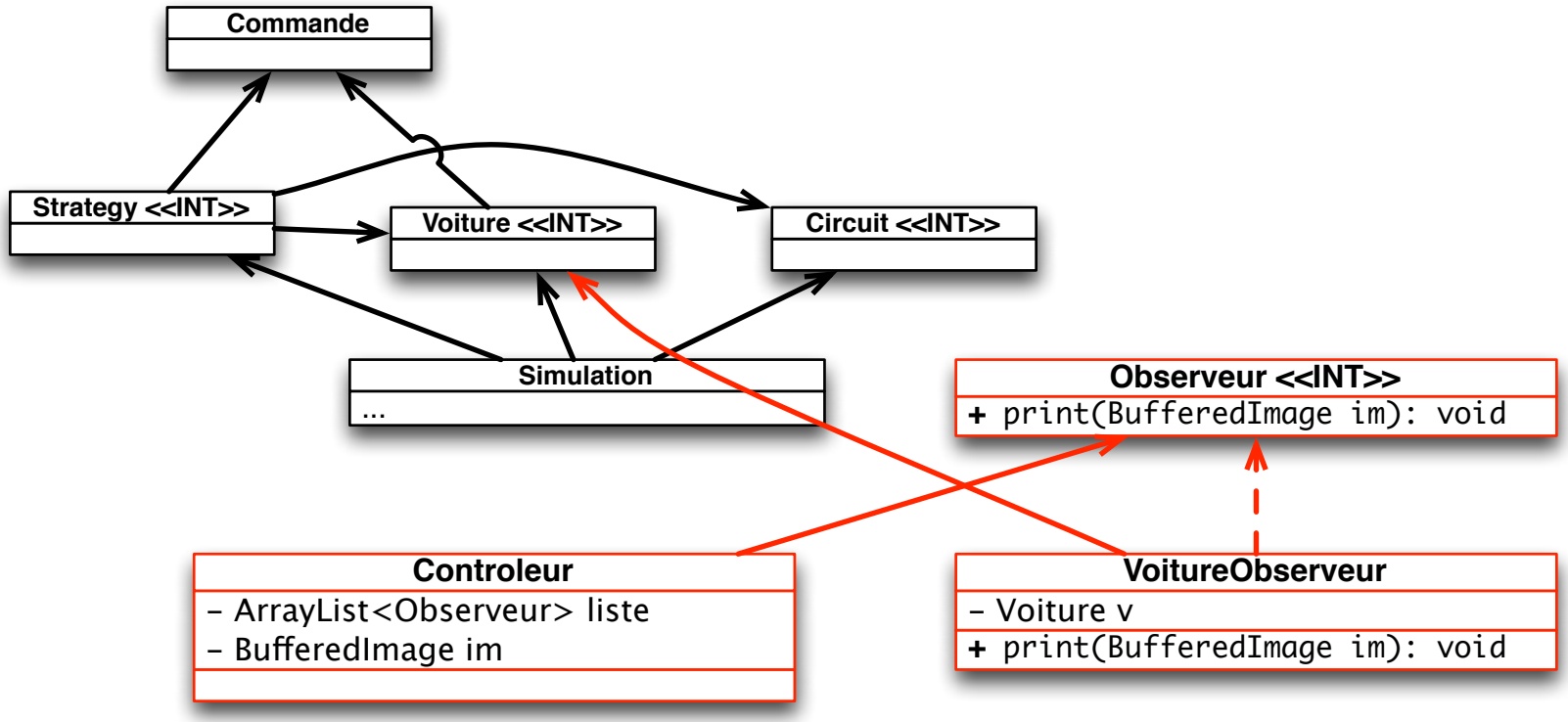
Proposition de base

- Information toujours stockées dans une image
- On travaille sur un système de branchement universel depuis le main
  - On pourra brancher un autre système plus tard
- Modèle physique :
  - Aucune référence à l'affichage
  - Modèle autonome : peut fonctionner sans affichage
- Visualisation
  - Branchement depuis le main



Observer

- Le code sera développé dans un nouveau package : `li260.view.observeurs`
- VoitureObserveur sait afficher la voiture dans l'image
- Le système d'affichage est découplé du modèle : on peut le débrancher ou en mettre un autre... Depuis le main



Principe général

Evènement *vs* programmation linéaire

Différents éléments réagissent les uns par rapport aux autres : ils émettent des événements et écoutent ce qui se passe autour d'eux.

Principe : chaque composant doit être indépendant, le main fait le lien entre certains émetteurs et certains récepteurs.

Exemple :

1. La voiture bouge, la simulation (physique) doit envoyer un signal : *update*
2. Si la vue est branchée, elle reçoit le message et procède à une mise à jour

Emetteur/récepteur

- **Émetteur** : la Simulation émet un évènement lorsqu'une mise à jour de l'affichage est nécessaire
  - L'émission revient à un appel de méthode dans le récepteur
  - L'émetteur doit connaître/stocker les récepteurs pour pouvoir leur envoyer le message
  - Les récepteurs doivent répondre à un cahier des charges : on doit être sur que la méthode de réception est implémentée...
- **Récepteur** : le Controleur recoit le message à travers un appel à la méthode manageUpdate(). Il met à jour l'image.
  - L'observeur fait le tampon entre la simulation et l'affichage, c'est le garant de l'indépendance

Cahiers des charges ⇒ utilisation d'interfaces

Définition des interfaces

- Emetteur :

```
1 public interface UpdateEventSender {
2     public void add(UpdateEventListener listener);
3     public void update();
4 }
```

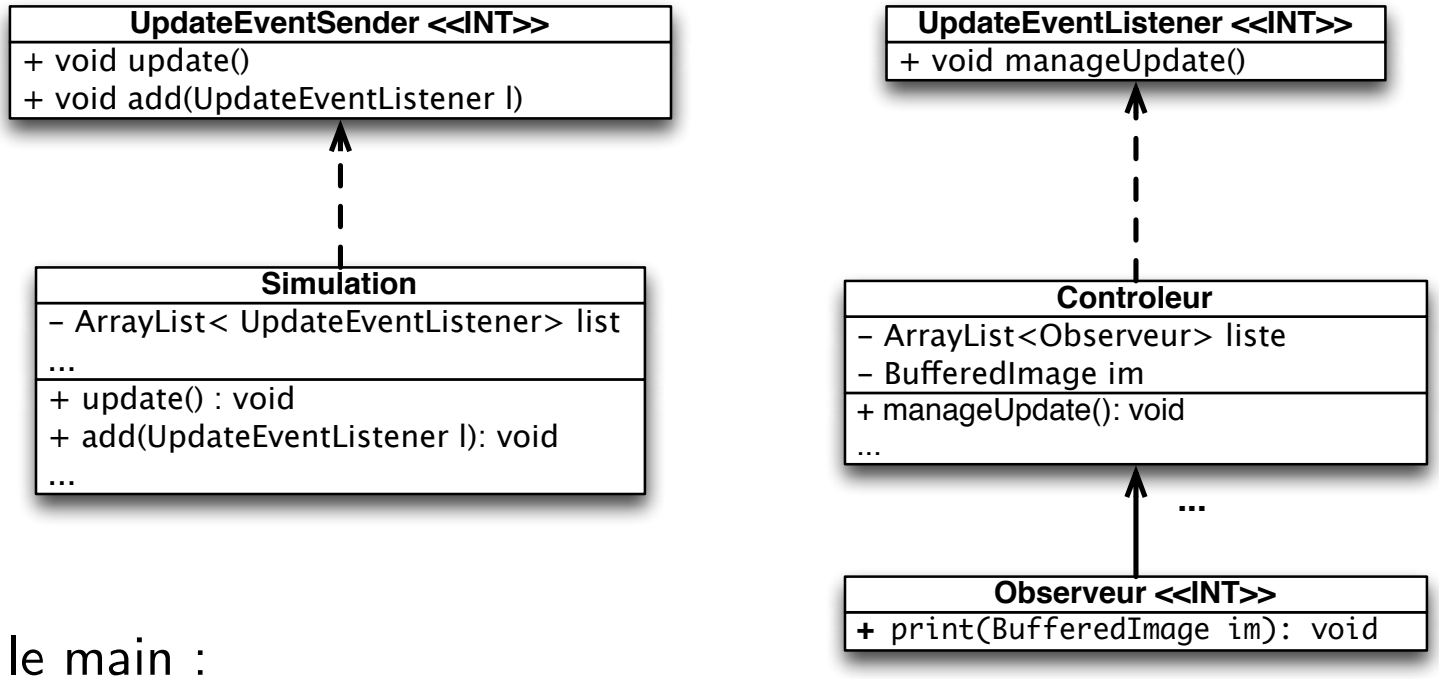
  1. Recevoir des écouteurs
  2. Leur envoyer un message

- Récepteur :

```
1 public interface UpdateEventListener {
2     public void manageUpdate();
3 }
```

  1. Réagir en cas de message

Emetteur/récepteur UML



Dans le main :

```
1  CircuitFactory cfac = new CircuitFactoryFromFile(filename);
2  Circuit track = cfac.build(); // si méthode non static
3  VoitureFactory vFac = new FerrariFactory(track);
4  Voiture v = vFac.build(); // si méthode non static
5  (...) // strategies
6  Controleur ihm = new Controleur(track);
7  ihm.add(new VoitureObserveur(v));
8  simu.add(ihm);
9  simu.play()
```

Séquence de communication

ACTE 0 :

- Le main fait le lien entre Vue(=Controleur dans cette modélisation) et Modèle physique
  (ligne 8 transparent précédent)

ACTE 1 : Emission/Diffusion

- La simulation décide d'envoyer un message
  appel à `void update()`

```
1 public void update() {
2   for(UpdateEventListener
3               listener: listeners)
4       listener.manageUpdate();
5 }
```

Séquence de communication (2)

ACTE 2 : Reception

Les méthodes `manageUpdate()` des récepteurs sont invoquées

ACTE 3 : Mise à jour de la vue

- Le controleur/vue a un attribut `ArrayList<Observer> liste`

```
1 public void manageUpdate() {
2   for(ObserveurImage o:liste)
3       o.print(image);
4 }
```

Evolutivité

Dans les cours à venir, nous étudierons :

- Bibliothèque SWING (interface graphique JAVA)
- Introduction à la 3D

Nouvelle notion : **affichage temps réel** (ou au moins dynamique)...

Bonne nouvelle :

L'architecture prévue est capable de gérer cela !
Rafraichissement de l'affichage lors des notifications de changements...

Limites

- Communication unidirectionnelle :
  - Controleur ⇒ Vue OK
  - Vue ⇒ Controleur KO !
- A-t-on besoin d'une telle communication ? Dans quel(s) cas ?

Limites

- Communication unidirectionnelle :
  - Controleur ⇒ Vue OK
  - Vue ⇒ Controleur KO !
- A-t-on besoin d'une telle communication ? Dans quel(s) cas ?
  - Mode pause, rectification de stratégie en direct...
- Comment coder ?

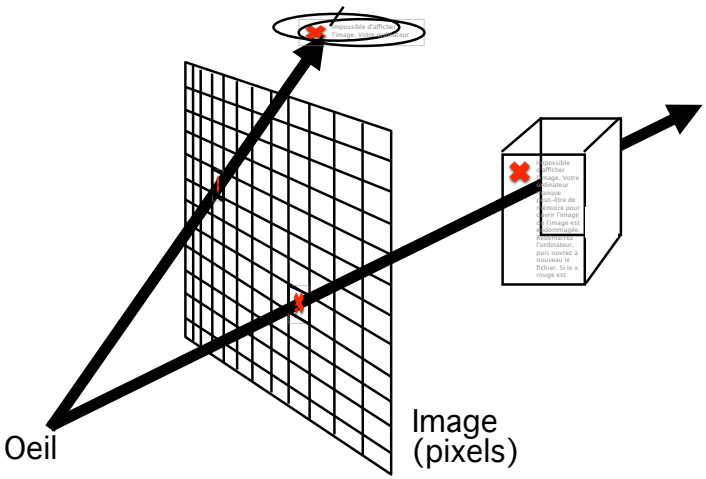
Limites

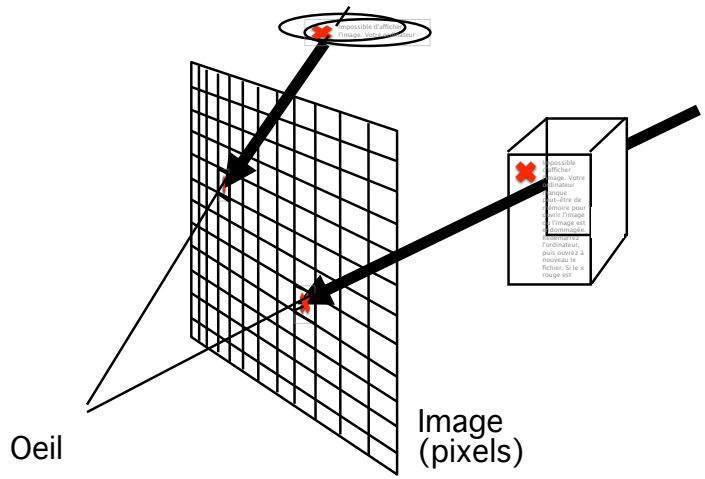
- Communication unidirectionnelle :
  - Controleur ⇒ Vue OK
  - Vue ⇒ Controleur KO !
- A-t-on besoin d'une telle communication ? Dans quel(s) cas ?
  - Mode pause, rectification de stratégie en direct...
- Comment coder ?
  - Créer un `PauseEvent` : le contrôleur est émetteur, le modèle récepteur...
  - Pas très générique... Mais facile et propre.

Quelques explications sur le fonctionnement

- Slides en partie tirés du cours de A. Meyer, Lyon 1
- Beaucoup de calcul matriciel (d'où les nouveaux usages des cartes graphiques)

2 approches duales en SI





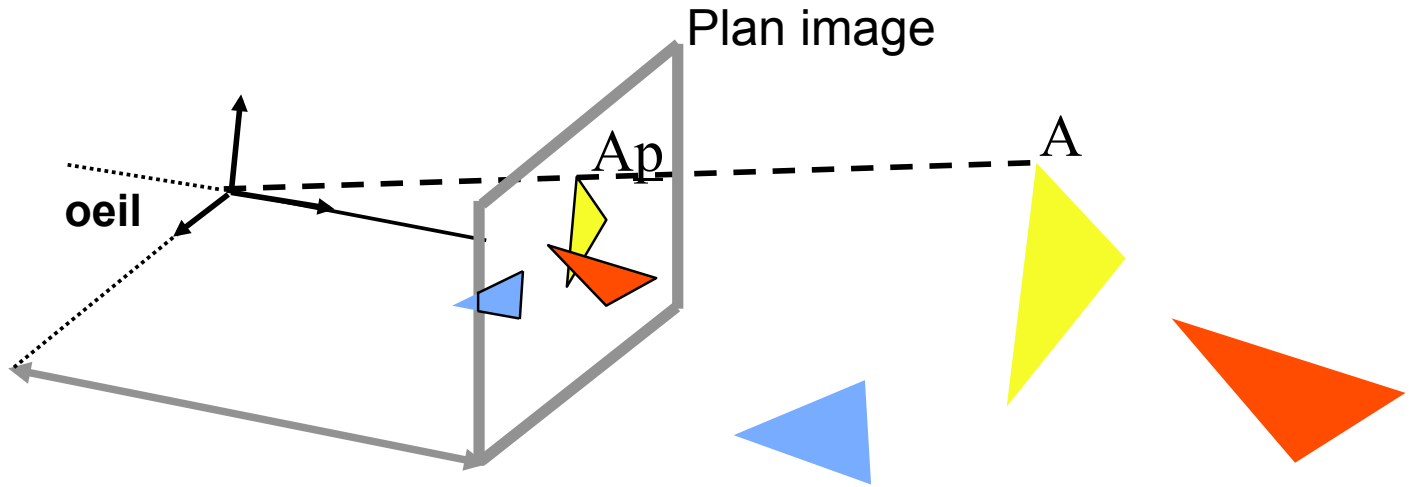
Des rayons sont lancés depuis l'œil vers la scène en passant par un pixel

Ray-tracing
- Image réaliste
- Lent

Les objets sont projetés sur l'écran dans la direction de l'œil.

Rendu projectif (cablé sur les cartes graphiques modernes -> temps réel)

Rendu projectif : PIPELINE

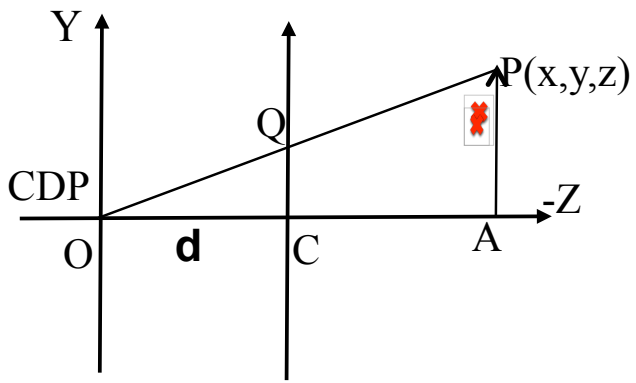


Pipeline

1. Clipping des polygones en 3D suivant la pyramide de vue
2. Projection des points sur le plan image
3. Remplissage des triangles (Rasterizing) dans l'image
   a. Suppression des parties cachées : Z-Buffer
   b. Calcul de la couleur : illumination

Projection perspective

- Besoin de perspective
- Configuration simple :



Plan image

$$\frac{CQ}{AP} = \frac{CO}{AO} \Leftrightarrow CQ = y' = \frac{y.d}{z}$$

$$x' = \frac{x.d}{z}$$

Si $d = 1 \Rightarrow y' = \frac{y}{z}$ et $x' = \frac{x}{z}$

d=distance focale

Matrice de projection : $M_{I \leftarrow C}$

$$M_{I \leftarrow C} = \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{pmatrix}$$

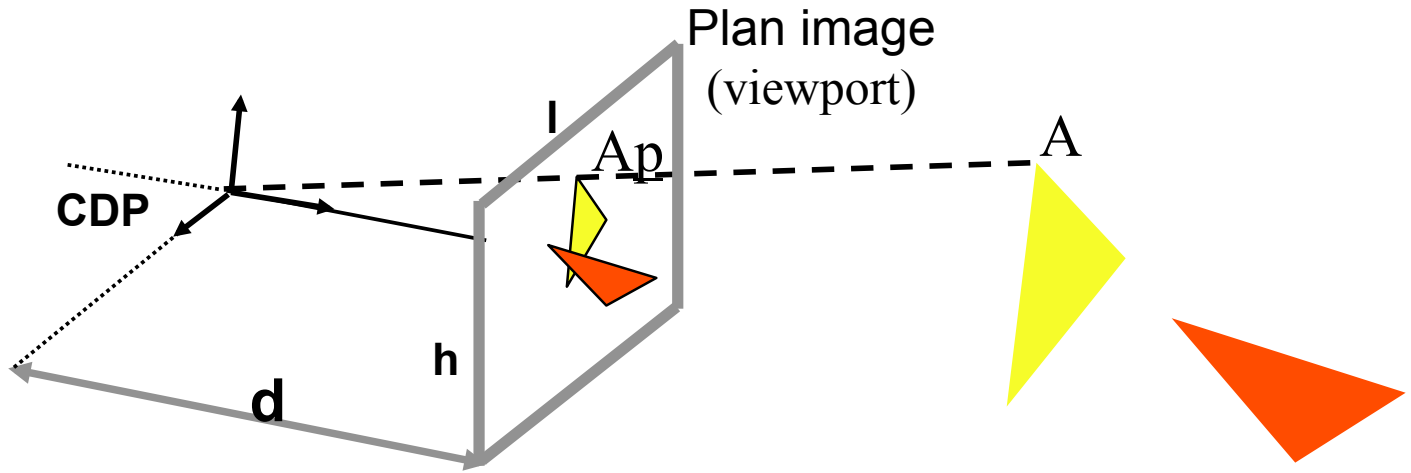
- Soit un point dans l'espace de la camera

$$(x, y, z, 1)$$

- Résultat : un point dans l'espace Image

$$(x, y, z, z/d) = \left(\frac{xd}{z}, \frac{yd}{z}, d, 1 \right)$$

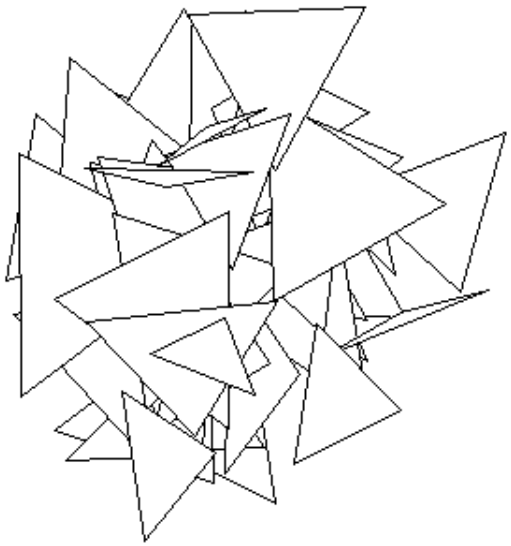
Rendu projectif

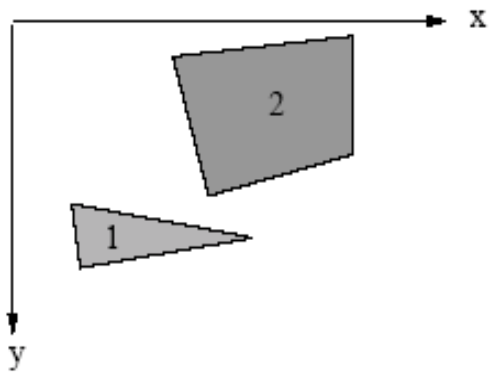


1. Projection des points sur le plan image
2. Clipping
3. Remplissage des triangles (rasterisation) dans l'image
5. Suppression des parties cachées
6. Calcul de la couleur : illumination

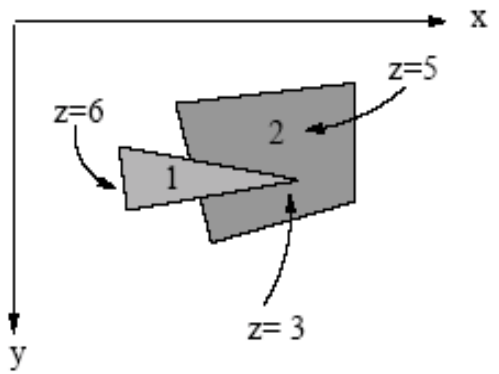
1ere idée : algo du peintre

Ambiguïtés

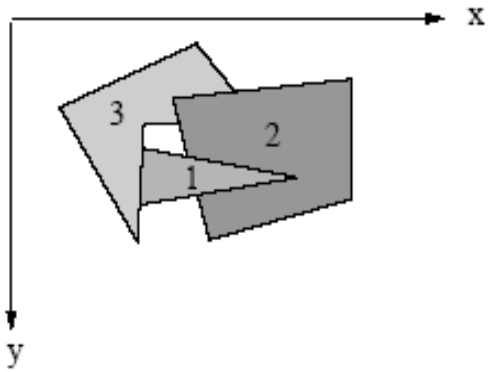




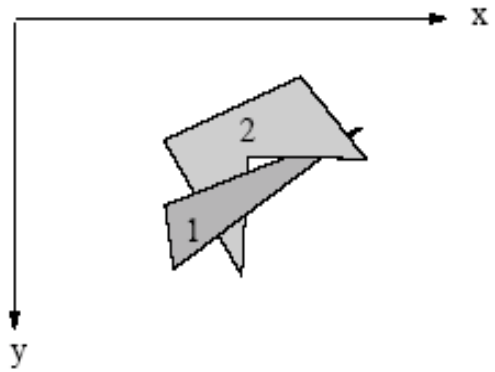
(a)



(b)



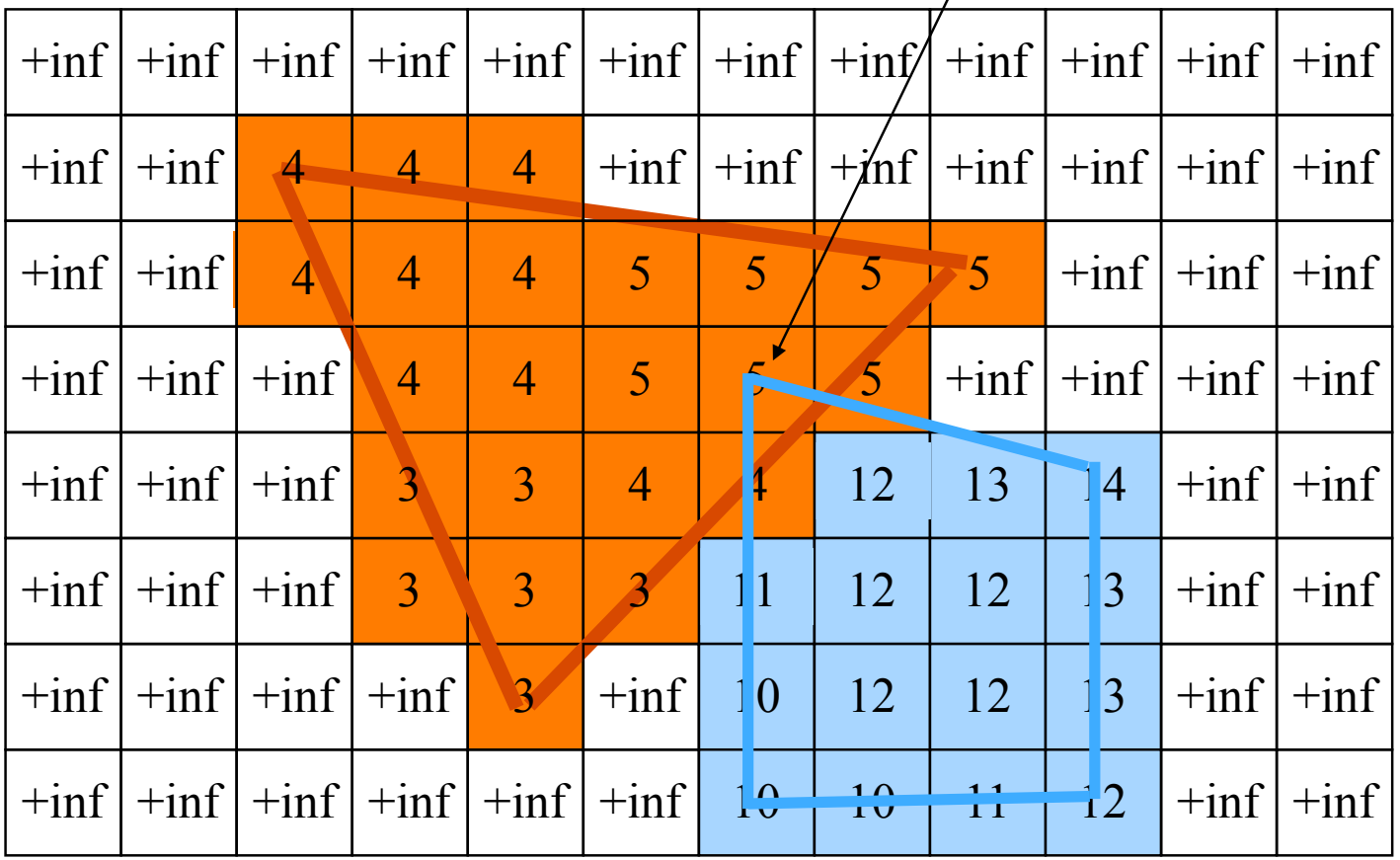
(c)



(d)

Z-Buffer

z=11>5 donc caché



OpenGL : par exemple

- Effacer le buffer et le zbuffer entre chaque image

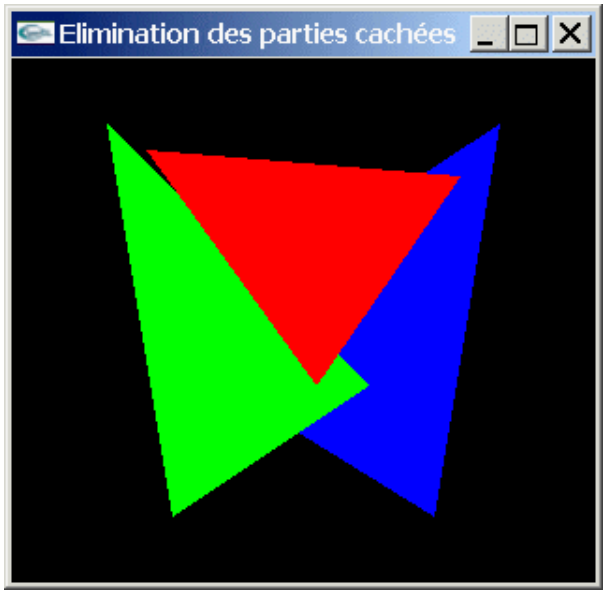
```
glClear(GL_COLOR_BUFFER_BIT|GL_DEPTH_BUFFER_BIT);
```

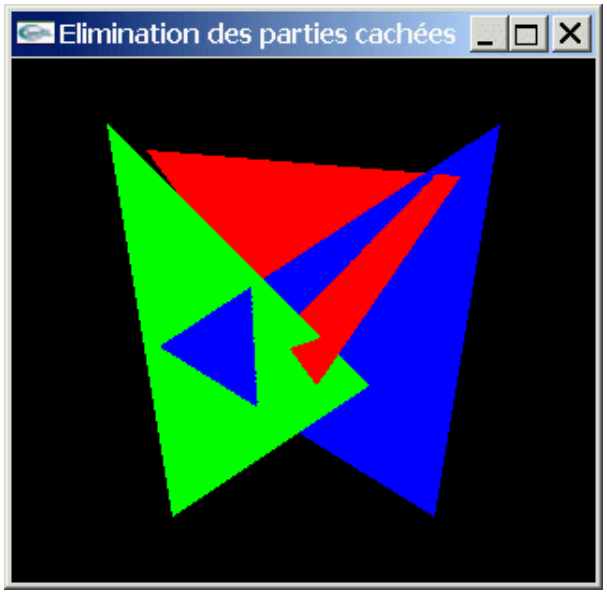
- Active le test des Z avec le Z-buffer

```
glEnable(GL_DEPTH_TEST) ;
```

activé

non activé





Python et 3D

- On propose d'utiliser pyglet
  - `pip install paquet -user -proxy=proxy.ufr-info-p6.jussieu.fr:3128`
- La puissance de l'architecture vient du fait qu'il n'y a pas grand chose à modifier pour faire de la 3D...

Gestion d'une fenêtre

```python
import pyglet
from OpenGL.GL import glLight
from pyglet.gl import *
from pyglet.window import key
from OpenGL.GLUT import *
from pyglet.image.codecs.png import PNGImageDecoder
```

Par extension d'un objet existant :

```python
class Window(pyglet.window.Window): # syntaxe de l'héritage
    xRotation = yRotation = 0
    increment = 5
    toDraw = []

    def __init__(self, width, height, title=''):
        super(Window, self).__init__(width, height, title)
        self.setup()
```

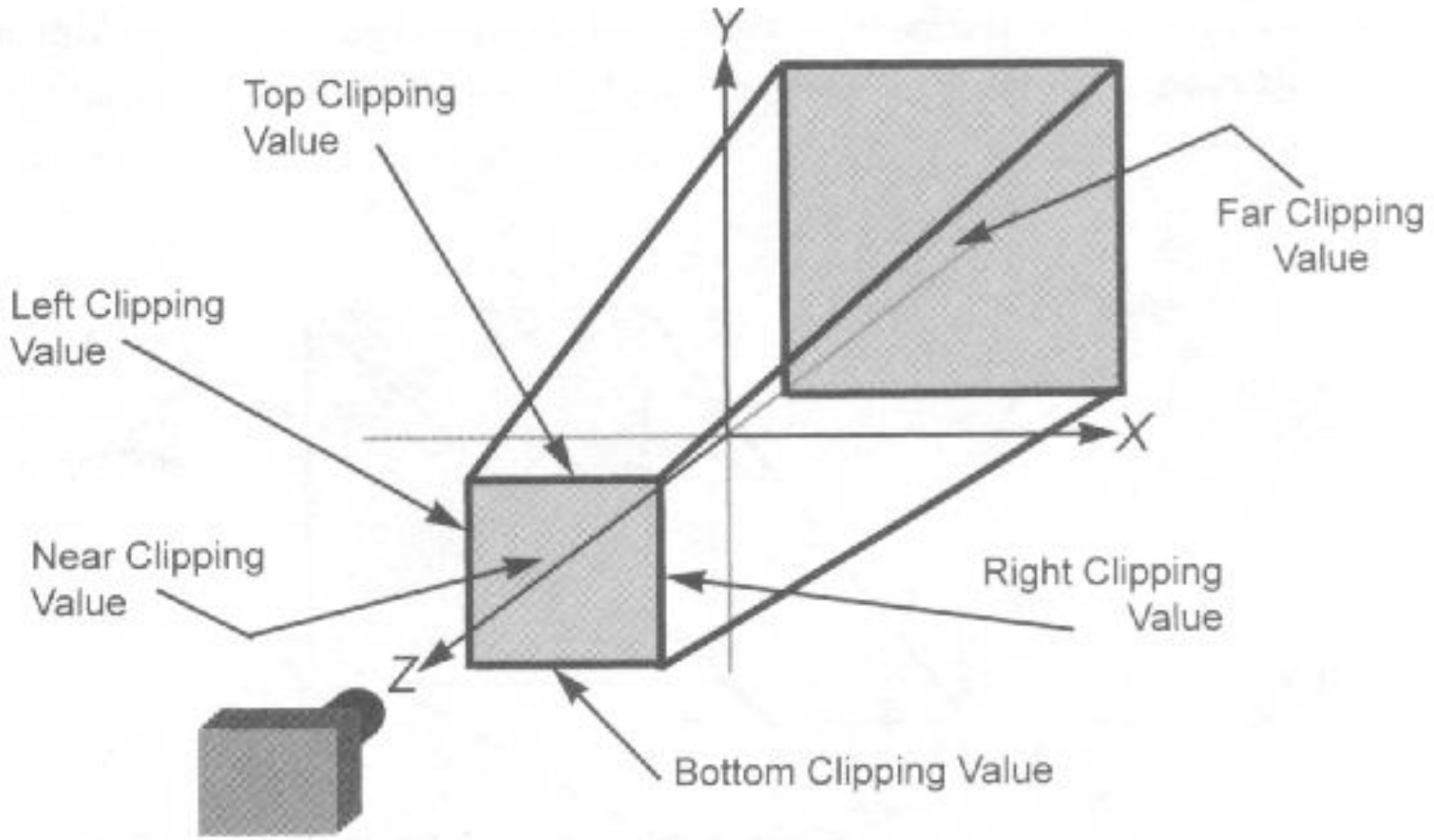
Setup

Réglages à vérifier :

- Tests de profondeur
- Définition de la scène

```
1  class Window(pyglet.window.Window): # syntaxe de l'héritage
2      ...
3      def setup(self):
4          # One-time GL setup
5          glClearColor(1, 1, 1, 1)
6          glColor3f(1, 0, 0)
7          glEnable(GL_DEPTH_TEST)
8          # using Projection mode
9          glViewport(0, 0, super(Window, self).width*2, \
10                 super(Window, self).height*2)  # taille de la scene
11         glMatrixMode(GL_PROJECTION)
12         glLoadIdentity()
13         # perspective
14         aspectRatio = super(Window, self).width / \
15                 super(Window, self).height
16         gluPerspective(35*self.zoom, aspectRatio, 1, 1000)
17         #
18         glMatrixMode(GL_MODELVIEW)
19         glLoadIdentity()
```

Définition de la fenêtre

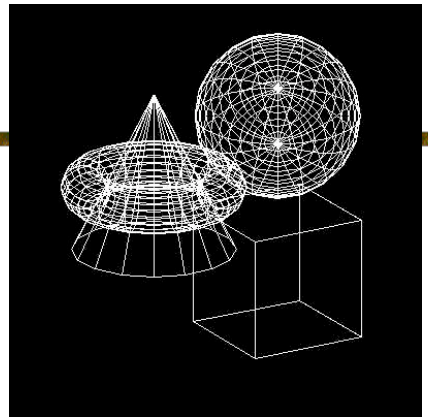


Option avancée

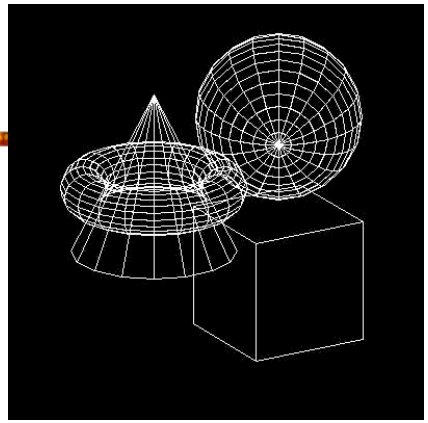
- Lumière, textures...

```
1    def setup_light(self):
2        # Simple light setup.  On Windows GL_LIGHT0 is enabled by default
3        # but this is not the case on Linux or Mac, so remember to alway
4        # include it.
5        glEnable(GL_LIGHTING)
6        glEnable(GL_LIGHT0)
7        glEnable(GL_LIGHT1)
8
9        # Define a simple function to create ctypes arrays of floats:
10       def vec(*args):
11           return (GLfloat * len(args))(*args)
12
13       glLightfv(GL_LIGHT0, GL_POSITION, vec(.5, .5, 1, 0))
14       glLightfv(GL_LIGHT0, GL_SPECULAR, vec(.5, .5, 1, 1))
15       glLightfv(GL_LIGHT0, GL_DIFFUSE, vec(1, 1, 1, 1))
16       glLightfv(GL_LIGHT1, GL_POSITION, vec(1, 0, .5, 0))
17       glLightfv(GL_LIGHT1, GL_DIFFUSE, vec(.5, .5, .5, 1))
18       glLightfv(GL_LIGHT1, GL_SPECULAR, vec(1, 1, 1, 1))
19
20       glMaterialfv(GL_FRONT_AND_BACK, GL_AMBIENT_AND_DIFFUSE, \
21            vec(0.5, 0.5, 0.5, 1))
22       glMaterialfv(GL_FRONT_AND_BACK, GL_SPECULAR, vec(1, 1, 1, 1))
23       glMaterialf(GL_FRONT_AND_BACK, GL_SHININESS, 50)
```

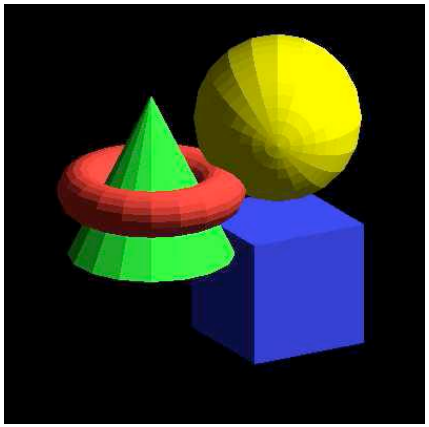
Modes de rendu



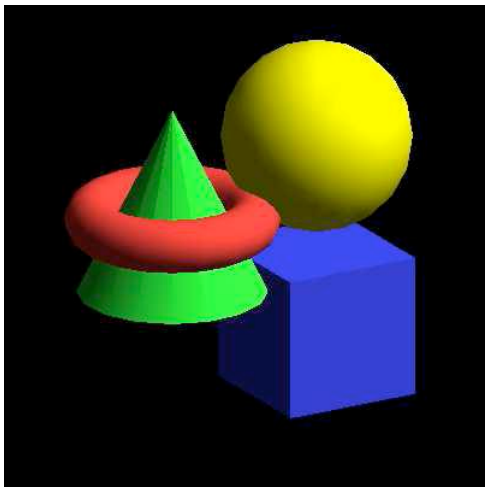
Fil de fer



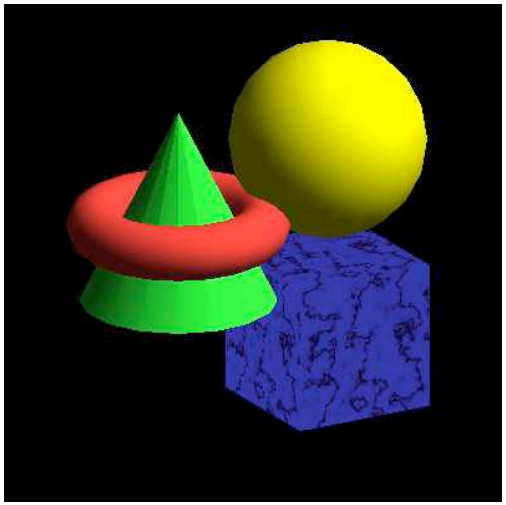
Faces cachées (objet)



Rendu Gouraud



Rendu Phong



Texture

Historique

Méthode d'affichage

- Conserver la structure générale
- Déporter l'affichage dans des éléments de base codés par ailleurs

```
1     def on_draw(self):
2         # Clear the current GL Window
3         self.clear()
4         self.set_camera() # cf plus tard
5         # Push Matrix onto stack
6         glPushMatrix()
7
8         glRotatef(self.xRotation, 1, 0, 0)
9         glRotatef(self.yRotation, 0, 1, 0)
10
11        for c in self.toDraw:
12            # Draw the six sides of the cube
13            c.draw()
14
15        # Pop Matrix off stack
16        glPopMatrix()
```

Dessinons !

```python
class Coord():
    def __init__(self,x,y,z,r,g,b):
        self.x=x
        self.y=y
        self.z=z
        self.r = r
        self.g = g
        self.b = b
        self.cote = 20

    def draw(self):
        glBegin(GL_QUADS)
        glColor3ub(self.r, self.g, self.b) # couleur
        glVertex3f(self.x, self.y, self.z) # point 1
        glVertex3f(self.x+self.cote, self.y, self.z) # point 2
        glVertex3f(self.x+self.cote, self.y+self.cote, self.z) # ...
        glVertex3f(self.x, self.y+self.cote, self.z)
        glEnd()
```

```python
if __name__ == '__main__':
    WINDOW = 400
    w= Window(WINDOW, WINDOW, 'Pyglet Colored Cube')
    w.toDraw += [ Coord(0,0,0, 100, 0, 0)]
    w.toDraw += [Coord(0, 40, 0, 0, 255, 0)]
    w.toDraw += [Coord(50, 0, 0, 0, 0, 255)]
    w.toDraw += [CoordTex(-50, 0, 0, 0, 255, 255, "matexture.png")]

    pyglet.app.run()
```

Interaction clavier

Gestion déjà prévue dans la fenêtre

```
1     def on_text_motion(self, motion):
2         if motion == key.UP:
3             self.xRotation -= self.increment
4         elif motion == key.DOWN:
5             self.xRotation += self.increment
6         elif motion == key.LEFT:
7             self.yRotation -= self.increment
8         elif motion == key.RIGHT:
9             self.yRotation += self.increment
10
11    def on_text(self, text):
12        print(self.zoom)
13        if text.find('z')>-1:
14            self.zoom *= 0.75
15        elif text.find('Z')>-1:
16            self.zoom *= 1.15
17        elif text.find('i')>-1:
18            pyglet.image.get_buffer_manager().get_color_buffer().\
19                save('screenshot.png')
```

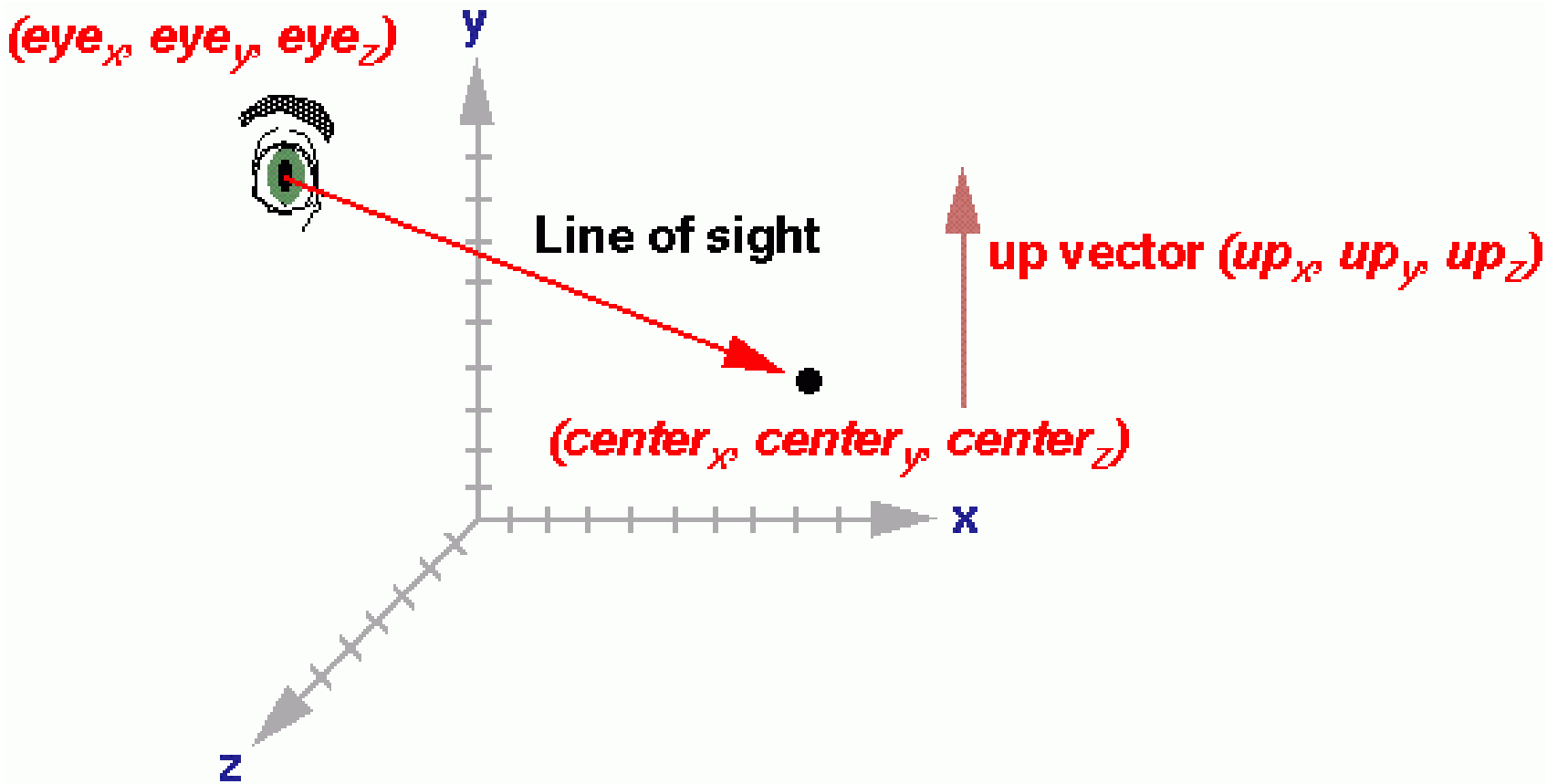
- Dernière ligne plus importante ! ! !

Fonctionnement de la caméra :

```
1 GLvoid gluLookAt( GLdouble eyex, GLdouble eyey,
2                   GLdouble eyez, GLdouble centerx, GLdouble
3                   centery, GLdouble centerz, GLdouble upx,
4                   GLdouble upy, GLdouble upz )
```

Fonctionnement de la caméra :

```
1 GLvoid gluLookAt( GLdouble eyex, GLdouble eyey,
2                   GLdouble eyez, GLdouble centerx, GLdouble
3                   centery, GLdouble centerz, GLdouble upx,
4                   GLdouble upy, GLdouble upz )
```



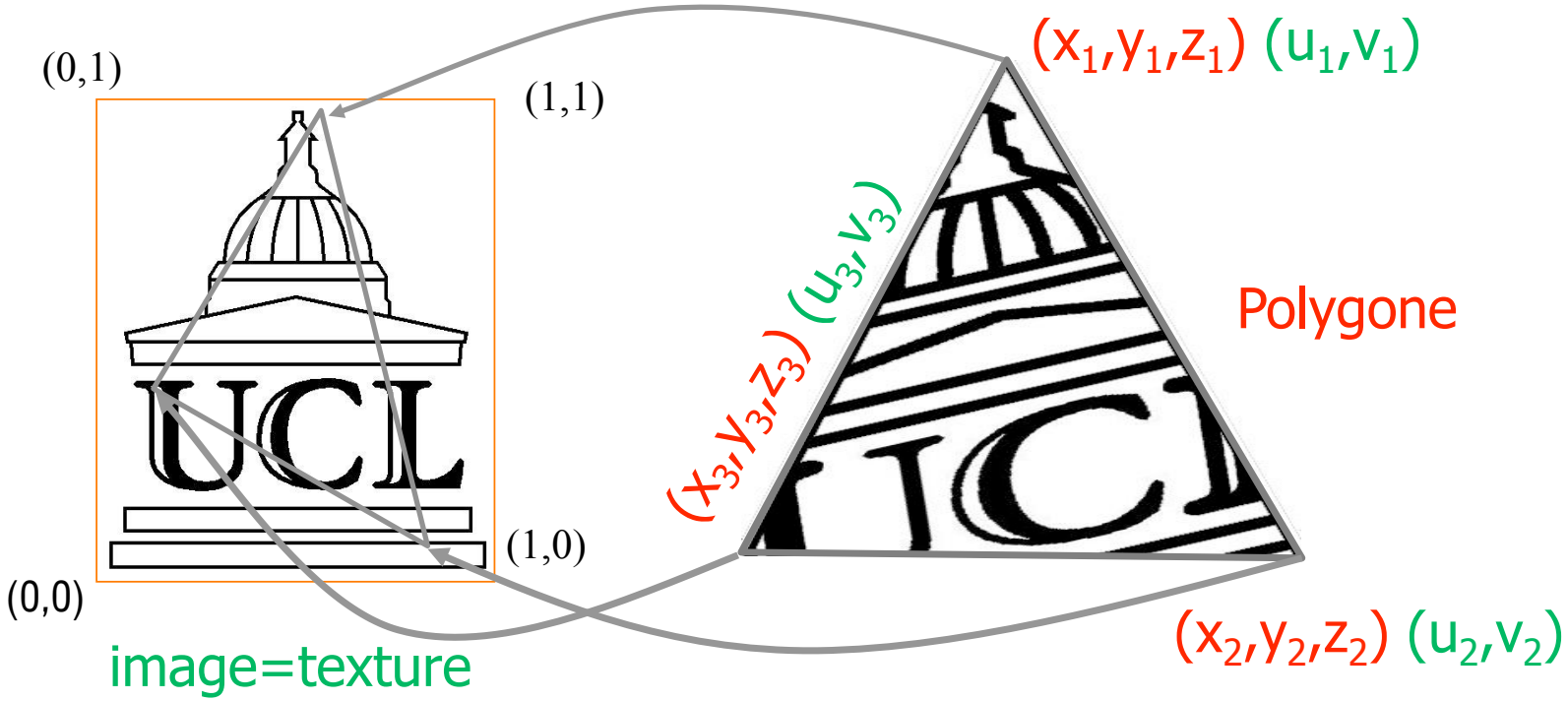
Mise en oeuvre de la caméra

```
1      def set_camera(self):
2          # using Projection mode
3          glViewport(0, 0, super(Window, self).width*2, super(Window, self
   # taille de la scene
4          glMatrixMode(GL_PROJECTION)
5          glLoadIdentity()
6          # perspective
7          aspectRatio = super(Window, self).width / super(Window, self).he
8          gluPerspective(35*self.zoom, aspectRatio, 1, 1000)
9          #
10         glMatrixMode(GL_MODELVIEW)
11         glLoadIdentity()
12         gluLookAt(-200, -200, 0, 200, 200, -100, 0, 0, 1)
```

Textures

Pour des rendus plus agréables et pour le terrain, il faut gérer les images en plus de la 3D :

Plaquage de la texture



- A chaque sommet de la face
  - Coordonnées textures (u,v)

Texture : application sur le cube

- texture = mémoire graphique + possède un identifiant
- Texture = image carrée, dim. en puissance de 2

```
1  class CoordTex(Coord):
2      def __init__(self,x,y,z,r,g,b, fname):
3          Coord.__init__(self, x, y, z, r, g, b)
4          im = pyglet.image.load(fname, decoder=PNGImageDecoder())
5          self.texture = im.get_texture()
6
7      def draw(self):
8          glBindTexture(self.texture.target, self.texture.id)
9          glPixelStorei(GL_UNPACK_ALIGNMENT, 1)
10
11         glBegin(GL_QUADS)
12         glTexCoord2f(0,0)
13         glVertex3f(self.x, self.y, self.z)
14         glTexCoord2f(1, 0)
15         glVertex3f(self.x+self.cote, self.y, self.z)
16         glTexCoord2f(1, 1)
17         glVertex3f(self.x+self.cote, self.y+self.cote, self.z)
18         glTexCoord2f(0, 1)
19         glVertex3f(self.x, self.y+self.cote, self.z)
20         glEnd()
21         glBindTexture(GL_TEXTURE_2D, 0)
```

Jouer avec les images

```
1  from PIL import Image
2  import random
3
4  if __name__ == '__main__':
5      # Image.load(filename)
6      d = 512
7      img = Image.new("RGB", (d, d), "white")
8      for i in range(d):
9          for j in range(d):
10             img.putpixel((i,j), (0,255,0))
11
12     for i in range(15000):
13         img.putpixel((random.randint(0,d-1), random.randint(0,d-1)), \
14           (random.randint(0,255), random.randint(0,255), random.randint(
15
16     for i in range(50):
17         for j in range(50):
18             img.putpixel((i+50,j+50), (255,0,0))
19
20     img.save("matexture.png", "png")
```

Résultat