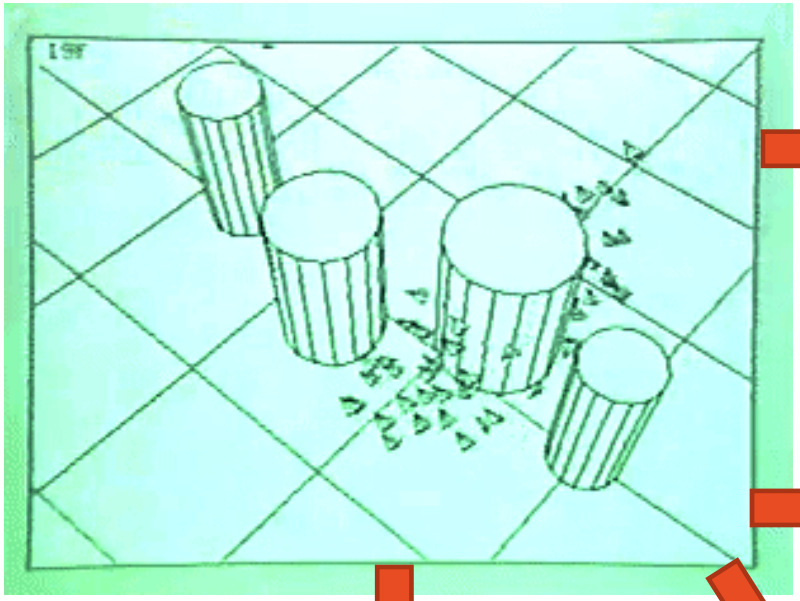
Image: © Alain Delorme

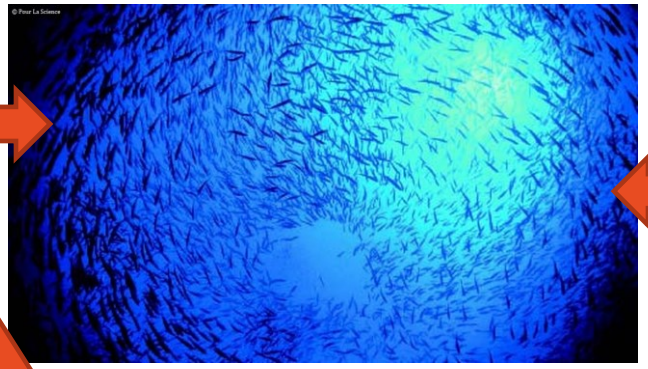
PROPAGATION D'ONDES DANS LES NUÉES DE BOIDS

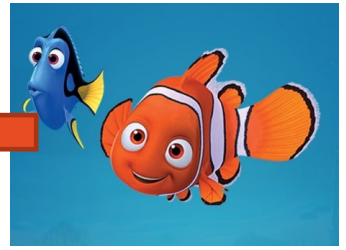
AMIEL Sacha
CONSTANTIN Leo
GLORIEUX Théo

INTRODUCTION

HISTOIRE

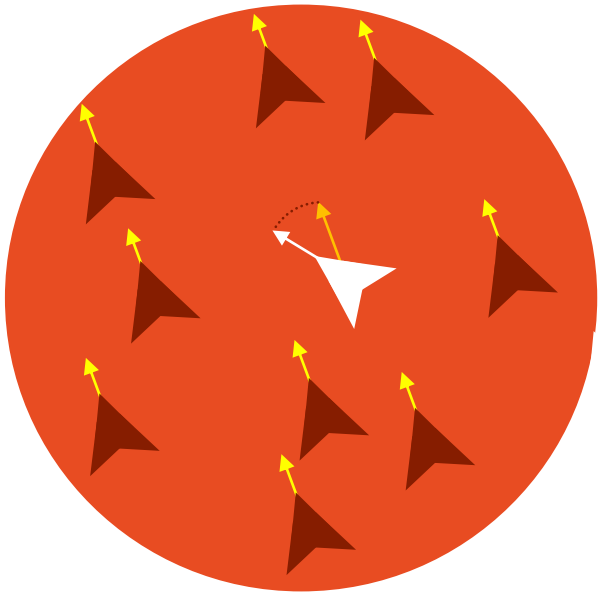




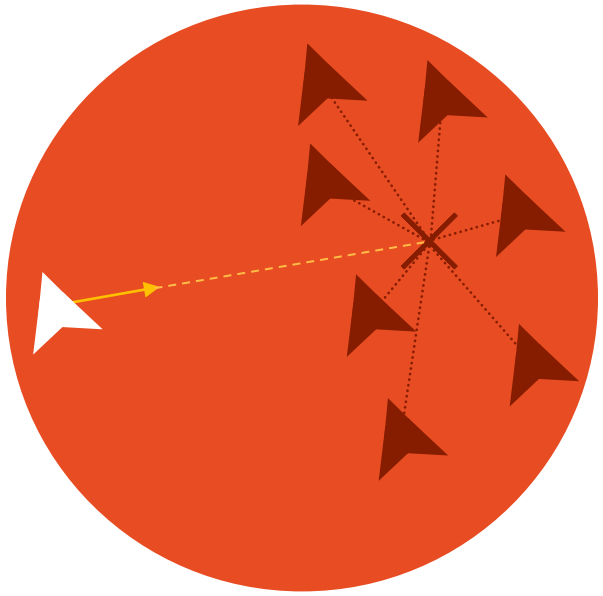


INTRODUCTION

LOIS



ALIGNEMENT



COHÉSION

SÉPARATION

EQUATIONS

Modèle *Vicsek*

$$\vec{x}_i(t+\mathrm{d}t) = \vec{x}_i(t) + \mathrm{d}t\vec{v}_i(t)$$

$$\vec{v}_i(t+\mathrm{d}t) = \vec{v}_i(t) + \mathrm{d}t\vec{F}_i + \vec{n}_i(t)$$

Bruit

$$\vec{F}_i := \vec{F}_\mathrm{a} + \vec{F}_{\mathrm{b}\cup\mathrm{c}} + \vec{F}_\mathrm{sc}$$

$$d_{ij} := ||\vec{x}_i - \vec{x}_j||$$

Distance

$$\vec{F}_\mathrm{a} := -A \sum_{d_{ij}<d_v} (\vec{v}_i - \vec{v}_j)$$

Alignement

$$\vec{F}_{\mathrm{b}\cup\mathrm{c}} := B \sum_j (\vec{x}_i - \vec{x}_j) \left((l_0/d_{ij})^3 - (l_0/d_{ij})^2 \right)$$

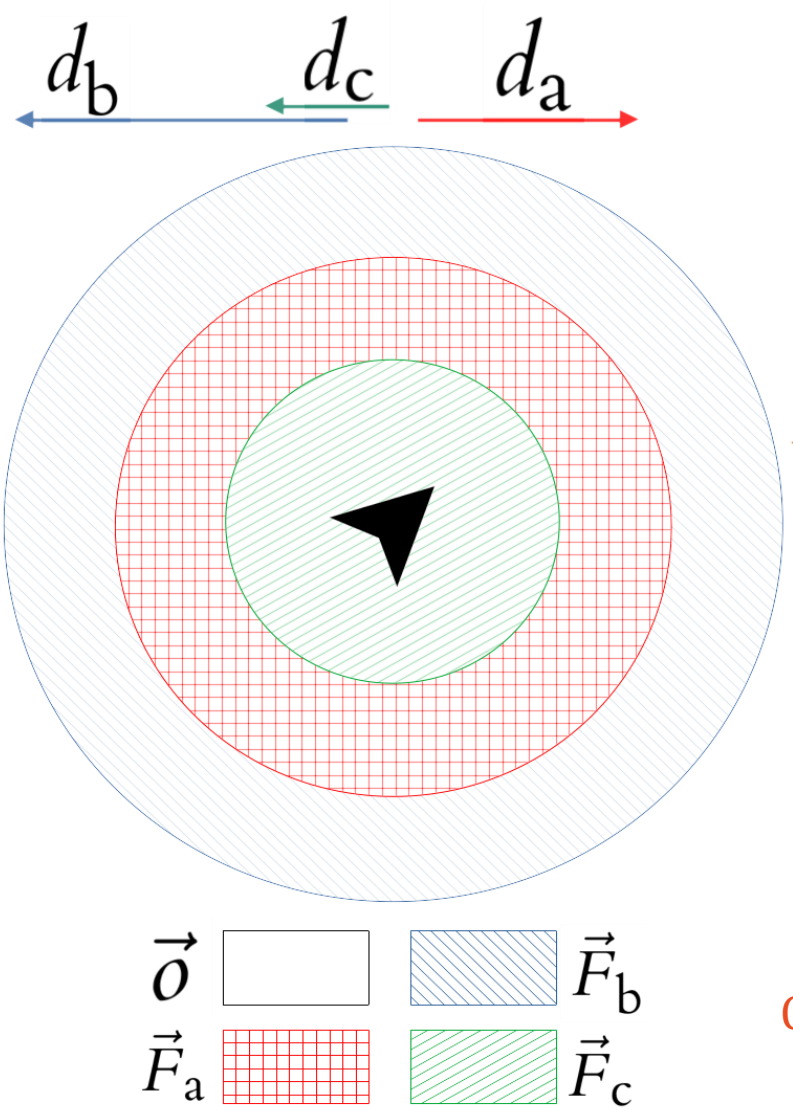
Cohésion & Séparation

$$\vec{F}_\mathrm{sc} := C\vec{v}_i \frac{v_0 - |\vec{v}_i|}{|\vec{v}_i|}$$

Préférence en vitesse

EQUATIONS

Modèle « maison »



$$\vec{x}_i(t+\mathrm{d}t) = \vec{x}_i(t) + \mathrm{d}t\,\vec{v}_i(t)$$

$$\vec{v}_i(t+\mathrm{d}t) = \vec{v}_i(t) + \mathrm{d}t\,\vec{F}_i + \vec{n}_i(t)$$

Bruit

$$\vec{F}_i \in \left\{\vec{F}_b, \vec{F}_a, \vec{F}_c, \vec{o}\right\}$$

$$d_{ij} := \left|\left|\vec{x}_i - \vec{x}_j\right|\right|$$

Distance

Séparation

$$\vec{F}_b := \frac{-B}{\#\{j \mid 0 < d_{ij} < d_b\}} \sum_{0<d_{ij}<d_b} \frac{\vec{x}_j - \vec{x}_i}{\|\vec{x}_j - \vec{x}_i\|}$$

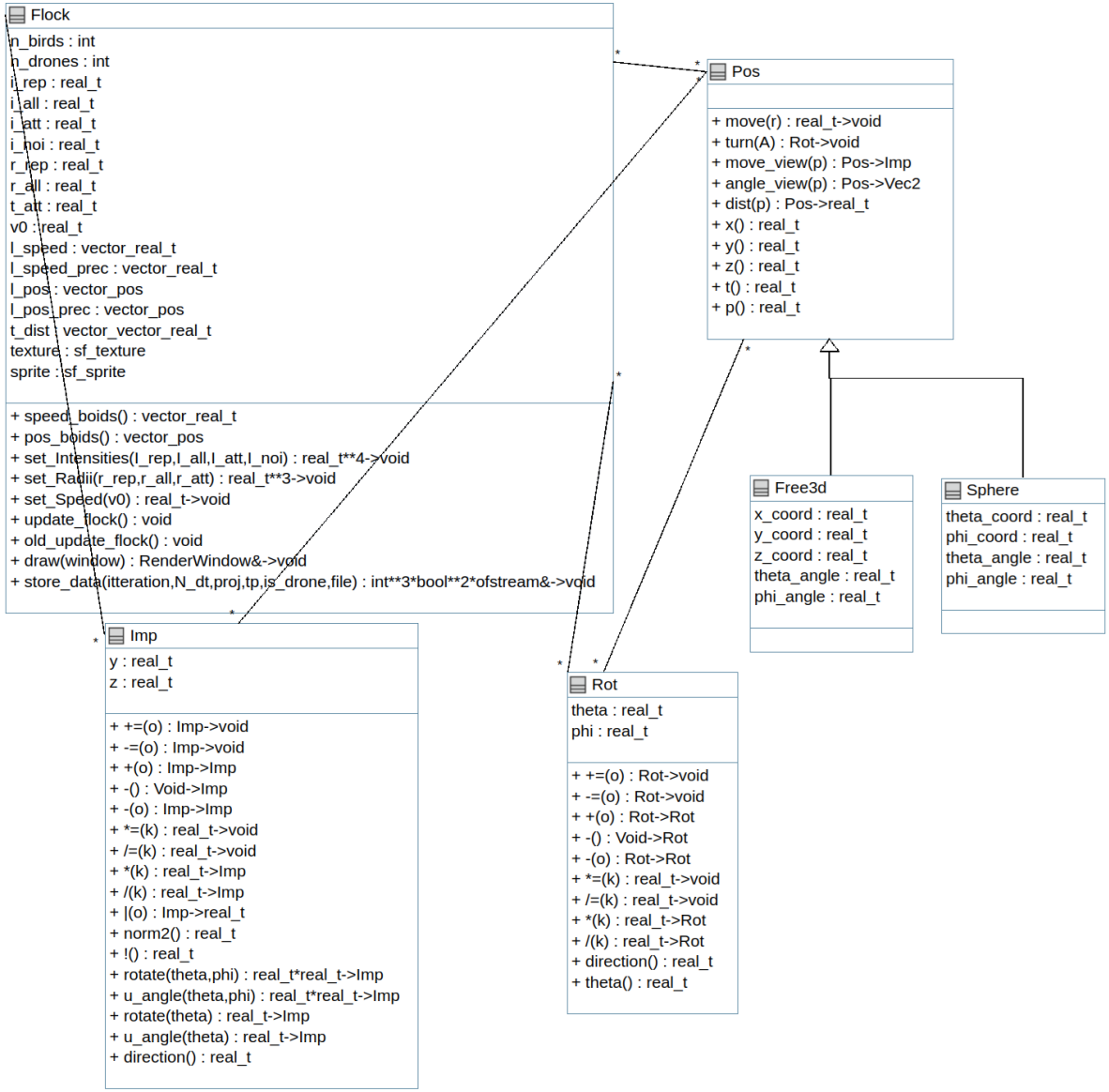
Alignement

$$\vec{F}_a := \frac{A}{\#\{j \mid d_b < d_{ij} < d_a\}} \sum_{d_b<d_{ij}<d_a} \frac{\vec{v}_j - \vec{v}_i}{\|\vec{v}_j - \vec{v}_i\|}$$

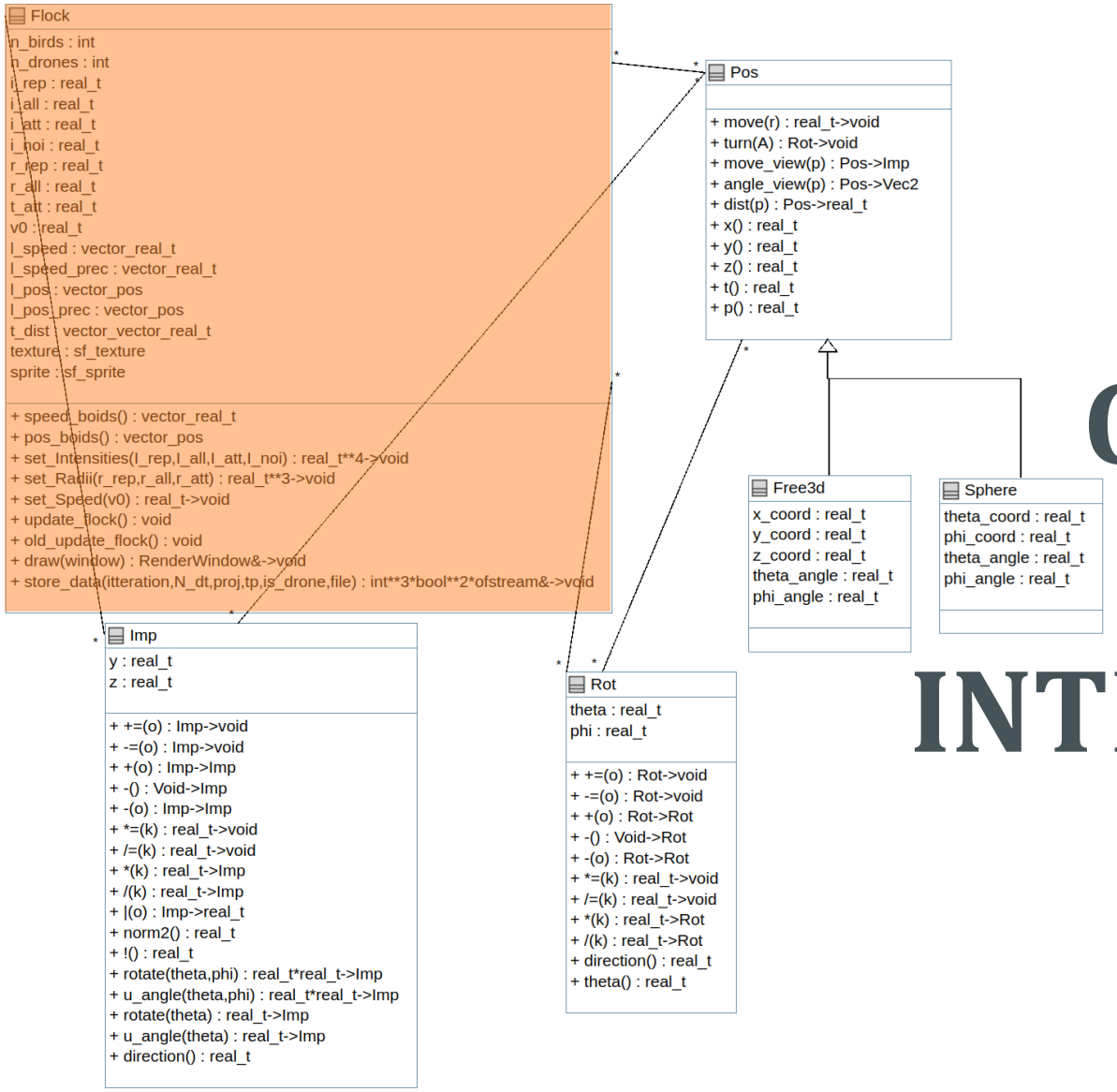
Cohésion

$$\vec{F}_c := C \frac{\vec{x}_j - \vec{x}_i}{\|\vec{x}_j - \vec{x}_i\|}\Bigg|_{j=\min\{j \mid d_a<d_{ij}<d_c\}}$$

STRUCTURE

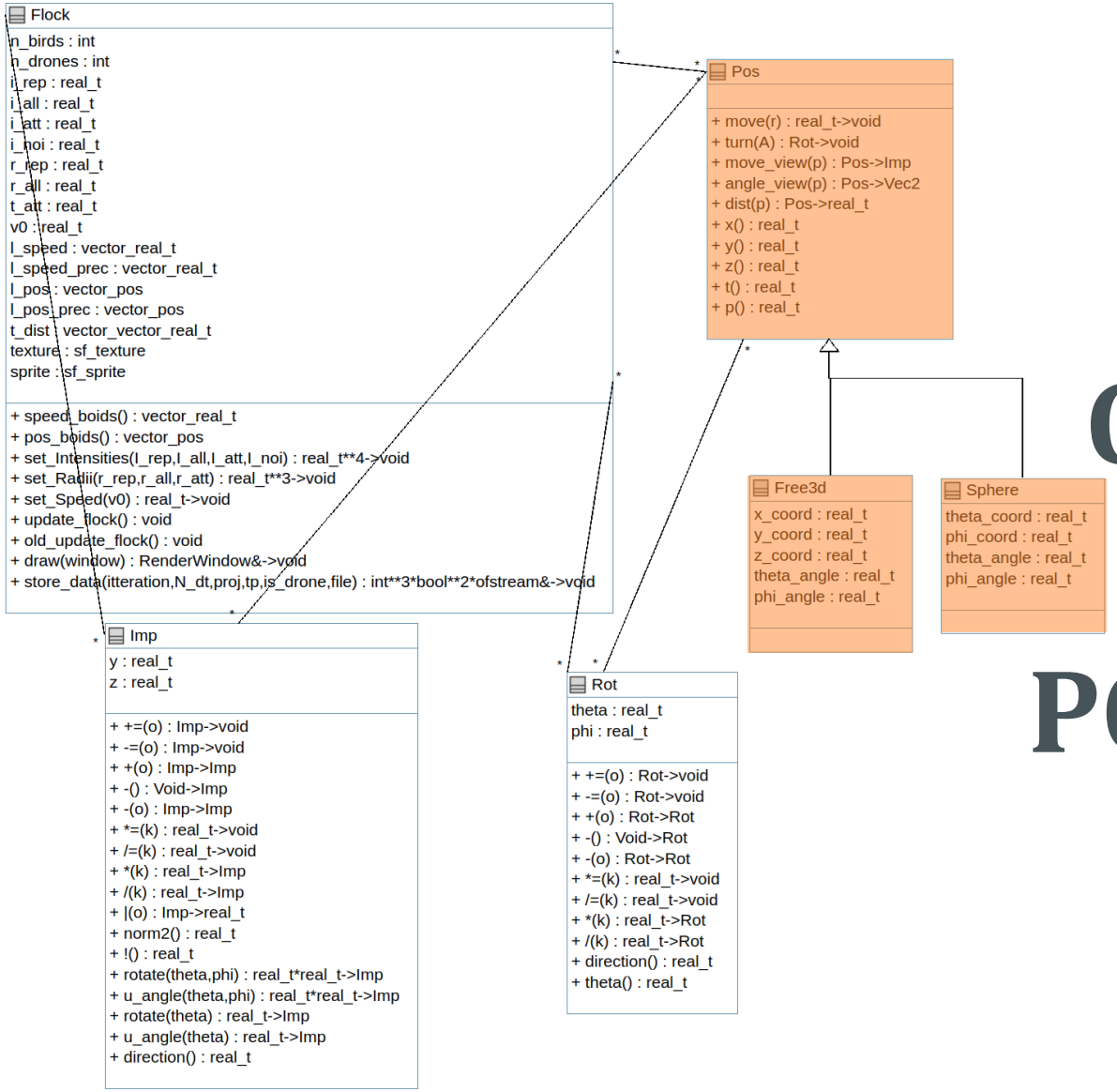

Flock
n_birds : int
n_drones : int
i_rep : real_t
i_all : real_t
i_att : real_t
i_noi : real_t
r_rep : real_t
r_all : real_t
t_att : real_t
v0 : real_t
l_speed : vector_real_t
l_speed_prec : vector_real_t
l_pos : vector_pos
l_pos_prec : vector_pos
t_dist : vector_vector_real_t
texture : sf_texture
sprite : sf_sprite
+ speed_boids() : vector_real_t
+ pos_boids() : vector_pos
+ set_Intensities(I_rep,I_all,I_att,I_noi) : real_t**4->void
+ set_Radii(r_rep,r_all,r_att) : real_t**3->void
+ set_Speed(v0) : real_t->void
+ update_flock() : void
+ old_update_flock() : void
+ draw(window) : RenderWindow&->void
+ store_data(itteration,N_dt,proj,tp,is_drone,file) : int**3*bool**2*ofstream&->void
Pos
+ move(r) : real_t->void
+ turn(A) : Rot->void
+ move_view(p) : Pos->Imp
+ angle_view(p) : Pos->Vec2
+ dist(p) : Pos->real_t
+ x() : real_t
+ y() : real_t
+ z() : real_t
+ t() : real_t
+ p() : real_t
Free3d
x_coord : real_t
y_coord : real_t
z_coord : real_t
theta_angle : real_t
phi_angle : real_t
Sphere
theta_coord : real_t
phi_coord : real_t
theta_angle : real_t
phi_angle : real_t
Imp
y : real_t
z : real_t
+ +=(o) : Imp->void
+ -=(o) : Imp->void
+ +(o) : Imp->Imp
+ -() : Void->Imp
+ -(o) : Imp->Imp
+ *=(k) : real_t->void
+ /=(k) : real_t->void
+ *(k) : real_t->Imp
+ /(k) : real_t->Imp
+ |(o) : Imp->real_t
+ norm2() : real_t
+ !() : real_t
+ rotate(theta,phi) : real_t*real_t->Imp
+ u_angle(theta,phi) : real_t*real_t->Imp
+ rotate(theta) : real_t->Imp
+ u_angle(theta) : real_t->Imp
+ direction() : real_t
Rot
theta : real_t
phi : real_t
+ +=(o) : Rot->void
+ -=(o) : Rot->void
+ +(o) : Rot->Rot
+ -() : Void->Rot
+ -(o) : Rot->Rot
+ *=(k) : real_t->void
+ /=(k) : real_t->void
+ *(k) : real_t->Rot
+ /(k) : real_t->Rot
+ direction() : real_t
+ theta() : real_t

STRUCTURE



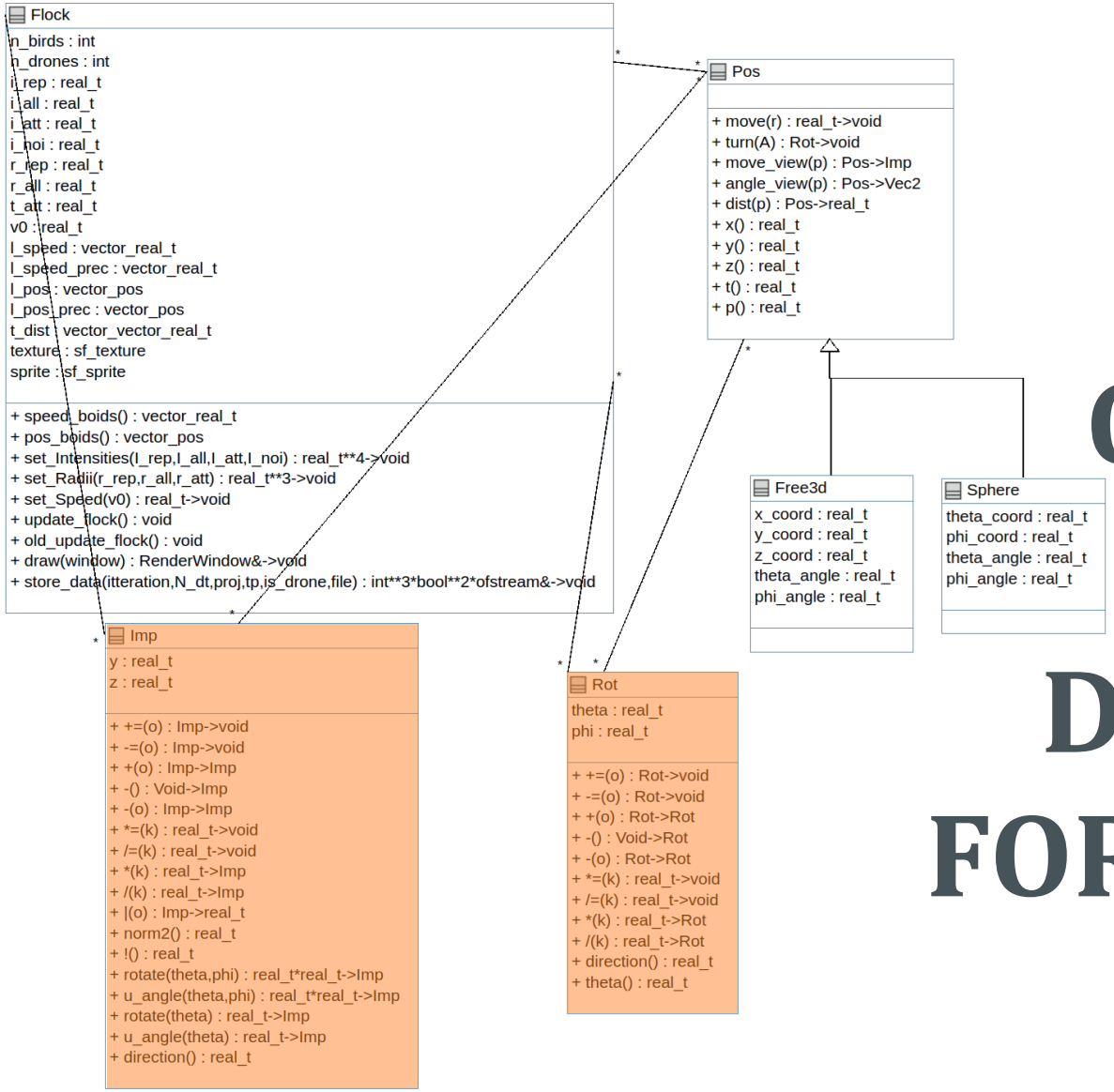
GESTION DES INTERACTIONS

STRUCTURE



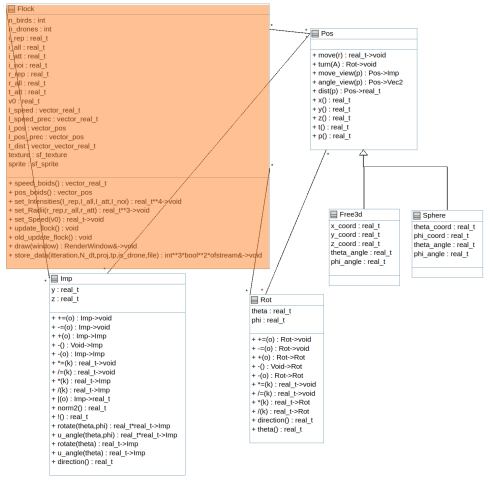
GESTION DES POSITIONS

STRUCTURE



GESTION DES DIALOGUE FORCE-ESPACE

FLOCK BOIDS



```
8   class Flock {
9       private :
10
13      int N_birds, n_drones;
14      real_t I_rep, I_all, I_att, I_noi;
15      real_t r_rep, r_all, r_att;
16      real_t v0;
21
22      std::vector<real_t>                l_speed;
23      std::vector<real_t>                l_speed_prec;
24      std::vector<pos>                   l_pos;
25      std::vector<pos>                   l_pos_prec;
26      std::vector<std::vector<real_t>> t_dist;
46
47      public :
51      sf ::Texture              texture;
52      std::vector<sf::Sprite> l_sprites;
56
57      std::vector<std::vector<real_t>> tab_dist    () const {return t_dist;}
58      std::vector<real_t>              speed_boids () const {return l_speed;}
59      std::vector<pos>                 pos_boids   () const {return l_pos;}
```

Individuelles:

- Position (et orientation)
- Vitesse

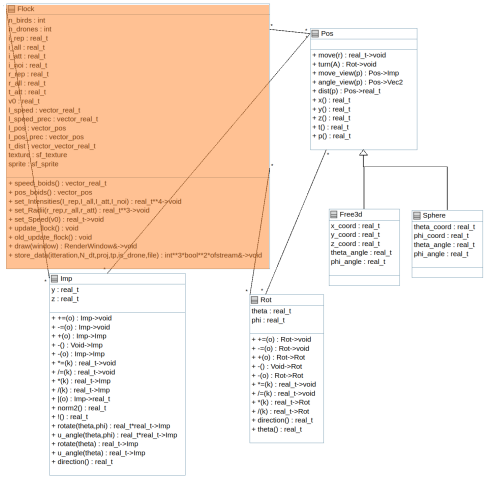
Globales:

- Propriétés graphiques
- Intensité (et autre propriétés) des forces

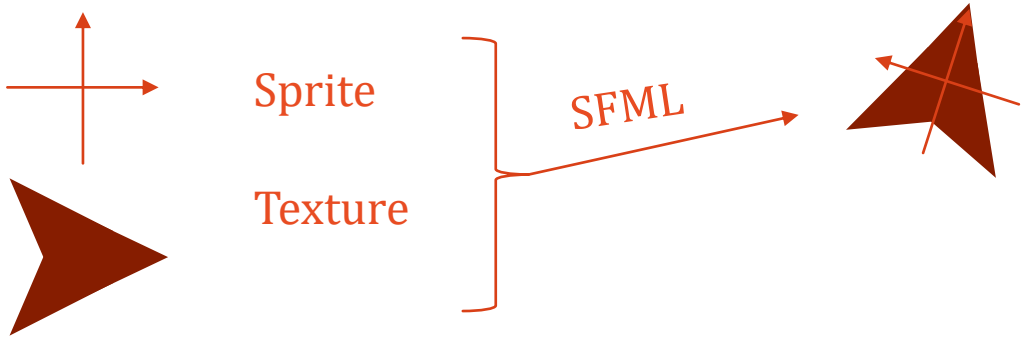
FLOCK

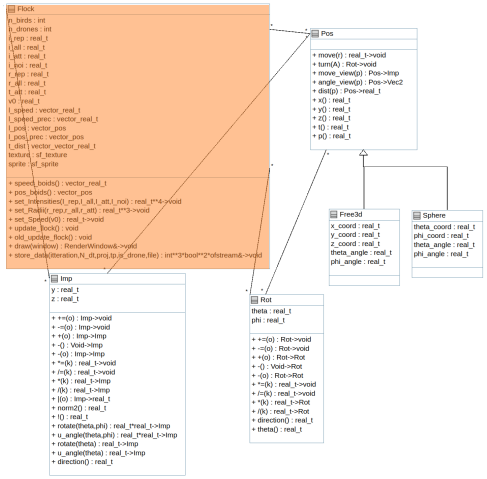
DRONES

```
void Flock::update_flock() {
    int i;
    int n= N_birds+n_drones;
    Imp F;
    l_pos_prec = l_pos;
    //l_speed_prec = l_speed;
    for (i=0; i<N_birds; i++){
        F = get_F (i);
        l_pos[i] ^ F.direction();
        l_pos[i] += v0;
        }
    for (i= N_birds; i<n; i++) {
        l_pos[i]+= l_speed[i];
    }
    update_dist();
```



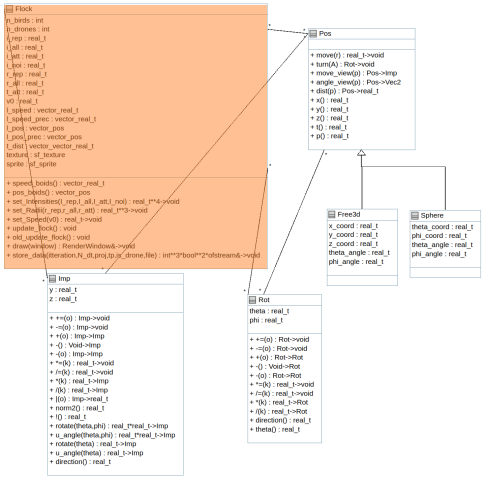
FLOCK GRAPHIQUE





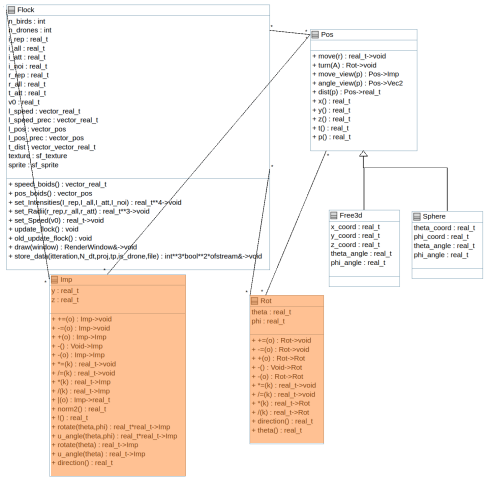

```
void Flock::update_graphics(){
    int i;
    int n= N_birds+n_drones;
    for (i=0; i<n; i++){
        l_sprites[i].setRotation(l_pos[i].p()*180/M_PI);
        l_sprites[i].setPosition(sf::Vector2f(l_pos[i].x()*0.9*WIDTH, l_pos[i].y()*0.9*HEIGHT));
    }
}
```

FLOCK GRAPHIQUE



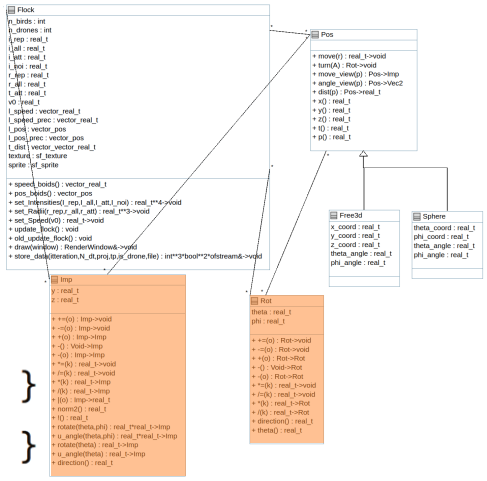

```
Flock::Flock (int N_birds_,real_t vi_birds,const std::vector<real_t>& speed_drones,const std::vector<pos>& pos_drones){
    N_birds=N_birds_;
    n_drones= speed_drones.size();
    I_all=0;
    v0=0;
    r_rep=0;
    I_att=0;
    r_all=0;
    I_rep=0;
    int n = N_birds_ +n_drones;
    texture.loadFromFile("oiseau.png");
    l_speed = std::vector<real_t> (n,vi_birds);
    l_pos = std::vector<pos> (n,pos(Imp{0,0,0}, Rot(0,0)));
    t_dist = std::vector<std::vector<real_t>> (n,std::vector<real_t> (n));
    l_sprites = std::vector<sf::Sprite> (n);
    int i;
    for (i=0;i<N_birds_;i++) {
        l_speed[i]= vi_birds;
        l_pos[i] = pos(Imp {dice(),dice(),dice()}, Rot (dice()*M_PI/180,dice()*M_PI/180));//dice()*M_PI/180));
        l_sprites[i].setTexture(texture);
        l_sprites[i].setColor(sf::Color(0, 0, 0, 108));//color,opacity
        l_sprites[i].setOrigin(sf::Vector2f(texture.getSize().x/2,texture.getSize().y/2));//to translate and rotate frm the center of the sprite
        l_sprites[i].setScale(0.07f,0.07f);//scale of the boid;
        l_sprites[i].setPosition(sf::Vector2f(l_pos[i].x()*0.9*WIDTH, l_pos[i].y()*0.9*HEIGHT));//initial position of the boid}
        l_sprites[i].setRotation(l_pos[i].p()*180/M_PI);//initial angle of the boid
    }
    for (i=N_birds; i<n;i++) {
        l_speed[i] =speed_drones[i-N_birds_];
        l_pos[i] = pos_drones[i-N_birds_];
        l_sprites[i].setTexture(texture);
        l_sprites[i].setColor(sf::Color(118, 56, 159, 128));//color,opacity
        l_sprites[i].setOrigin(sf::Vector2f(texture.getSize().x/2,texture.getSize().y/2));//to translate and rotate frm the center of the sprite
        l_sprites[i].setScale(0.09f,0.09f);//scale of the boid;
        l_sprites[i].setPosition(sf::Vector2f(l_pos[i].x()*0.9*WIDTH, l_pos[i].y()*0.9*HEIGHT));//initial position of the boid}
        l_sprites[i].setRotation(l_pos[i].p()*180/M_PI);//initial angle of the boid
    }
    l_speed_prec = l_speed;
    l_pos_prec = l_pos;
    update_dist();
}
```

IMP & ROT




```
class Rot {
public :
  void theta(real_t valeur) {p_theta=valeur;sync_interval();}
  void phi(real_t valeur) {p_phi=valeur; sync_interval();}
  real_t theta() const {return p_theta;}
  real_t phi () const {return p_phi;}
  Rot(const Rot & autre) = default;
  Rot(real_t theta_, real_t phi_) {p_theta= theta_; p_phi=phi_; sync_interval();}
  Rot() {p_theta=0;p_phi=0;}
  void operator+=  (Rot o)         { theta(p_theta + o.theta()); phi(p_phi + o.phi()); }
  void operator-=  (Rot o)         { theta(p_theta - o.theta()); phi(p_phi - o.phi()); }
  Rot operator+   (Rot o)   const { return Rot(p_theta+o.p_theta, p_phi+o.p_phi); }
  Rot operator-   ()         const { return Rot( -p_theta, -p_phi ); }
  Rot operator-   (Rot o)   const { return Rot( p_theta-o.p_theta, p_phi-o.p_phi ); }
  void operator*=  (real_t k)       { theta(p_theta * k); phi(p_phi * k); }
  void operator/=  (real_t k)       { theta(p_theta/k); phi(p_phi/k); }
  Rot operator*   (real_t k) const { return Rot( k*p_theta, k*p_phi ); }
  Rot operator/   (real_t k) const { return Rot( p_theta/k, p_phi/k); }
private :
  real_t p_theta, p_phi;
  void sync_interval();
};
```

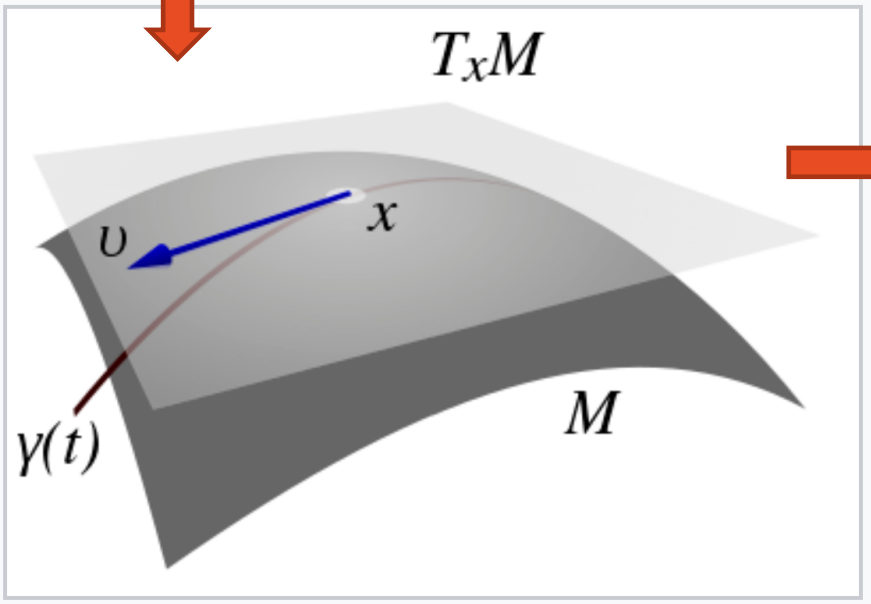
IMP & ROT



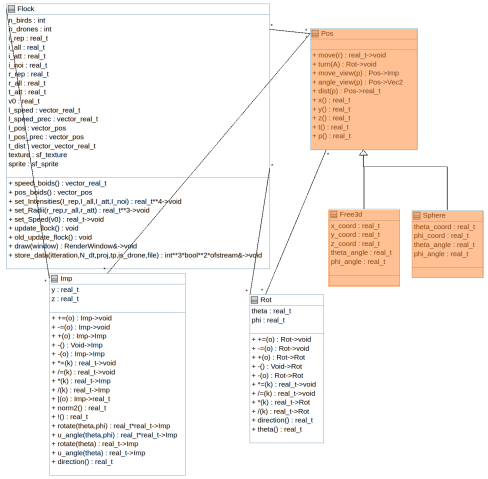

```
struct Imp {
  real_t x, y,z;
  void operator+=  (Imp o)         { x += o.x; y += o.y; z+=o.z; }
  void operator-=  (Imp o)         { x -= o.x; y -= o.y; z-=o.z; }
  Imp operator+   (Imp o)   const { return Imp{ x+o.x, y+o.y, z+o.z }; }
  Imp operator-   ()         const { return Imp{ -x, -y, -z }; }
  Imp operator-   (Imp o)   const { return Imp{ x-o.x, y-o.y, z-o.z }; }
  void operator*=  (real_t k)       { x *= k; y *= k; z*=k; }
  void operator/=  (real_t k)       { x /= k; y /= k; z/=k; }
  Imp operator*   (real_t k) const { return Imp{ k*x, k*y, k*z }; }
  Imp operator/   (real_t k) const { return Imp{ x/k, y/k, z/k }; }
        real_t operator| (Imp o)   const { return x*o.x + y*o.y+z*o.z ; }
        real_t norm2     ()         const { return x*x + y*y +z*z; }
        real_t operator! ()         const;
  Rot direction() const;
  Imp  rotate     (Rot angles2) const;
  static Imp u_angle (Rot angles);
};
```

POSITION

EQUATIONS *LOCALES*



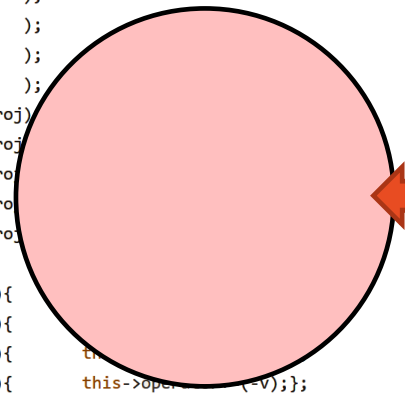
The tangent space $T_x M$ and a tangent vector $v \in T_x M$, along a curve traveling through $x \in M$.

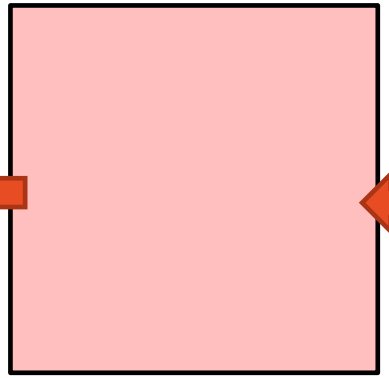


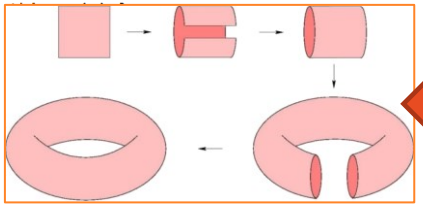

```
1   class position {
2    public:
3     void   move       (real_t  v   );
4     void   move       (Imp     v   );
5     void   turn       (Rot     a   );
6     real_t dist       (position Y   );
7     Imp    move_view  (position Y   );
8     Rot    angle_view (position Y   );
9     real_t x          (int     proj);
10    real_t y          (int     proj);
11    real_t z          (int     proj);
12    real_t t          (int     proj);
13    real_t p          (int     proj);
```

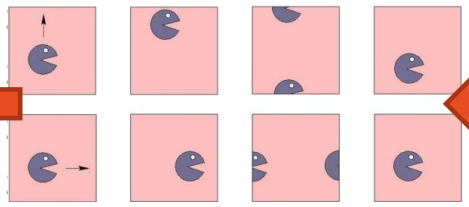
POSITION

```cpp
#ifndef _annulus_H_
#define _annulus_H_
#include "../Vec/Vec.h"

class annulus {
 public:
  annulus (Imp X, Rot A);

  void   operator+= (real_t  v   );
  void   operator+= (Imp     v   );
  void   operator^  (Rot     a   );
  real_t operator|  (annulus Y   );
  Imp    operator-  (annulus Y   );
  Rot    operator<< (annulus Y   );
  real_t x          (int     proj);
  real_t y          (int     proj
  real_t z          (int     proj
  real_t t          (int     proj
  real_t p          (int     proj

  void   move       (Imp     v){
  void   move       (real_t  v){
  void   operator-= (Imp     v){
  void   operator-= (real_t  v){      this->oper                (-v);};
  void   turn       (Rot     a){      this->operator^ ( a);};
  real_t dist       (annulus Y){return this->operator| ( Y);};
  Imp    move_view  (annulus Y){return this->operator- ( Y);};
  Rot    angle_view (annulus Y){return this->operator<<( Y);};
  Rot    operator>> (annulus Y){return  Y<<(this[0]);};
  real_t x (){return this->x(0);};
  real_t y (){return this->y(0);};
  real_t z (){return
  real_t t (){return
  real_t p (){return

 private:
  real_t x_coord;
  real_t y_coord;
  real_t t_angle; // stands for θ, angle with axis 0.
};
```

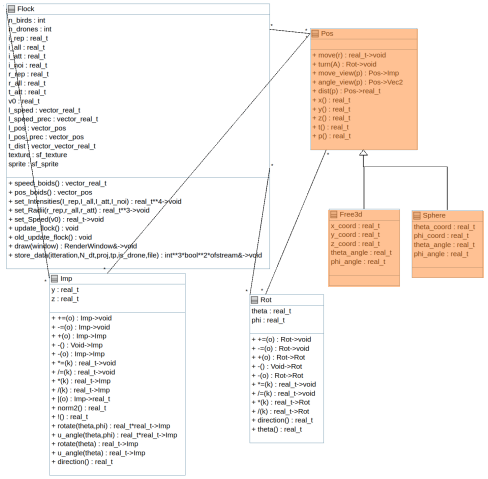








```cpp
#ifndef _POSITION_H_
#define _POSITION_H_

/*
#include "free2d.h"
using pos = free2d;
*/

/*
#include "annulus.h"
using pos = annulus;
*/
```



$[0,1]^2$

$\mathbb{R}^2$ $\mathbb{R}^3$

$(\mathbb{R}/\mathbb{Z})^2$

université PARIS-SACLAY

PROPAGATION D'ONDES DANS LES NUÉES DE BOIDS

AMIEL Sacha
CONSTANTIN Leo
GLORIEUX Théo