Rapport de projet d'informatique graphique

Avatar

BARASCUD Julien – PRAT Constantin

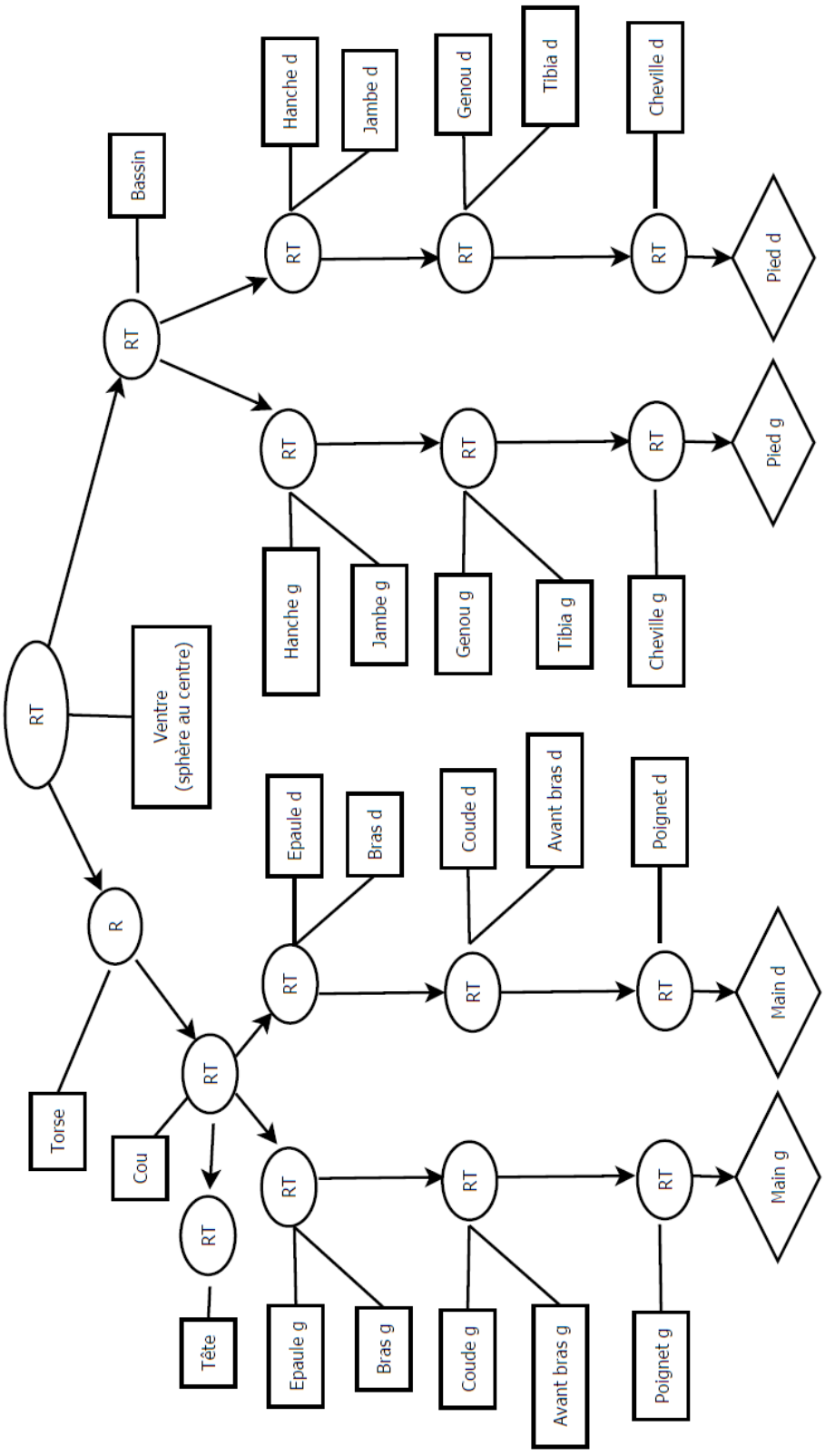
Sommaire:

Introduction:

Avant même d'avoir eu le sujet du projet, nous avions pour but de créer un avatar beaucoup plus complexe que celui, basique, du TD. Nous l'avions trouvé inesthétique, et peu proche d'un homme réel. Nous avons débuté par dessiner l'avatar sur papier, pour ensuite tracer notre graphe de scène. Ce premier jet comportait déjà pratiquement tous les raffinements au niveau esthétique présent sur notre projet final. En cours de route nous avons cependant changé radicalement l'aspect de notre avatar, pour le transformer en Link, personnage principal des jeux vidéo Zelda.

Source : www.cosplayhouse.com

I) Graphe de scène



II) Fonctionnalités de base

A- Aspect esthétique

Afin d'avoir une possibilité de mouvements plus réaliste, notre avatar présente de nombreuses articulations:

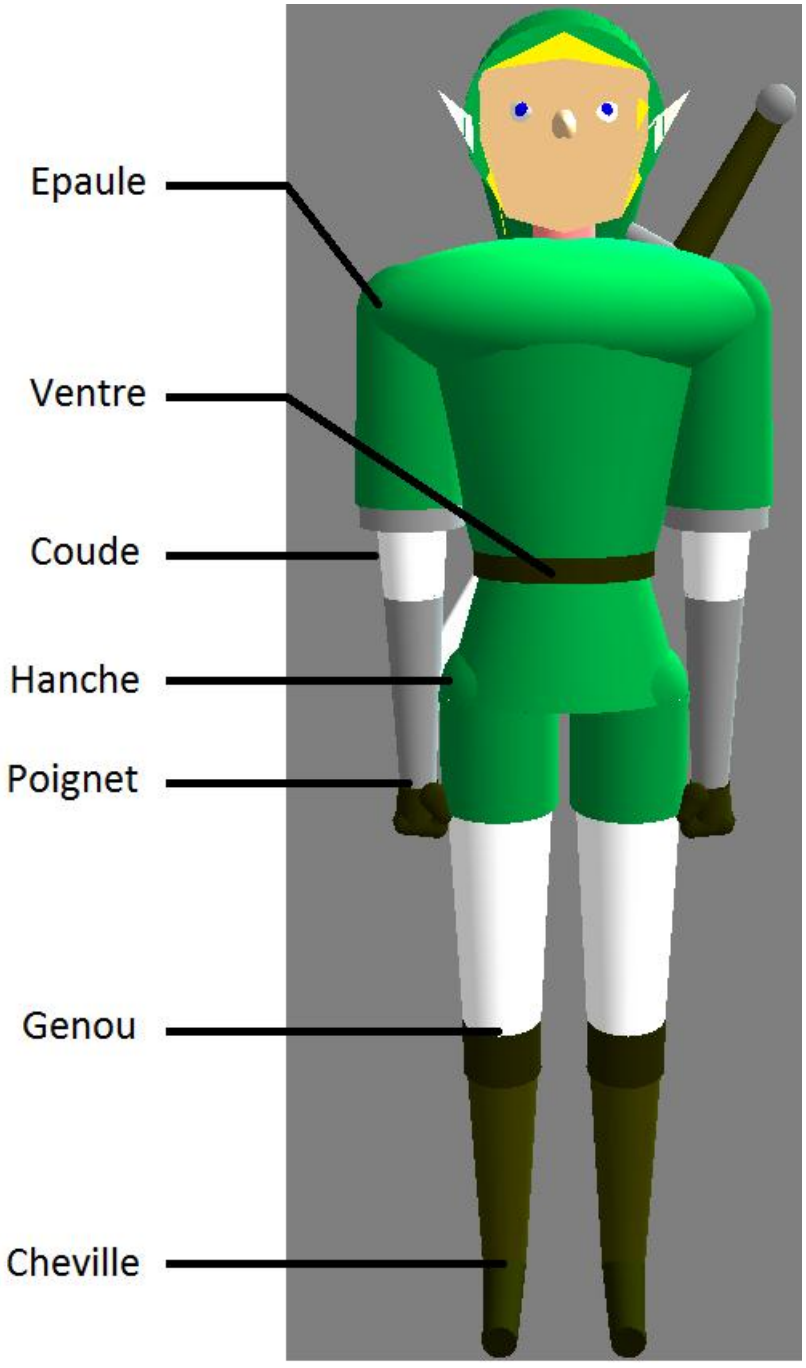


Schéma : Articulations

Toutes ces articulations sont définies dans notre fichier structures.h, dans la structure T_mon_avatar, ainsi que toutes les longueurs caractéristiques (pied, tibia, cuisse, bassin, torse, cou, tête, bras, avant-bras et épée) et les rayons caractéristiques (bas et haut du torse, ventre, hanche, genou, cheville, épaule, coude, poignet, cou, tête et bassin).

Cela permet une modification rapide de la taille de tous les membres de l'avatar sans pour autant avoir à changer l'intégralité du code. Cela pourrait en effet devenir gênant et fastidieux étant donné les animations et la gestion des collisions que nous avons réalisées, points sur lesquels nous reviendrons plus tard.

Afin d'améliorer encore notre avatar, nous lui avons donné la forme du personnage du jeu vidéo Zelda, Link. Cet elfe porte notamment un chapeau très reconnaissable, car ayant une forme particulière. Afin de retrouver cette forme, nous avons créé une succession de cylindres peu hauts, en effectuant une légère rotation et une translation entre chaque cylindre. Plutôt que de garder une tête ronde, nous avons modelé le visage de toutes pièces à l'aide de triangles. Les oreilles pointues de Link ont également été reproduites à l'aide de triangles.

Le corps a été façonné de manière à rester le plus fidèle au modèle original. Les articulations sont toutes modélisées par des sphères, et les membres par des cylindres. Le torse est quant à lui constitué d'une sphère et d'un cylindre. Nous avons aplati ceux-ci à l'aide de la fonction glScalef pour coller le plus fidèlement possible à la réalité d'un humain.

Pour un réalisme plus important, nous avons décidé de créer des mains, non articulées, à notre personnage. Bien que non articulées, elles possèdent chacune 5 doigts placés de manière à se conformer au plus proche à un poing fermé.

Nous avons décidé d'ajouter une épée à notre avatar, pour coller encore plus au personnage. La lame a été construite avec un simple cylindre et cône, et l'utilisation de la fonction glScalef pour l'aplatir.

Nous avons défini une structure T_angle pour chaque articulation.

```
typedef struct angle {
   float xmin;
   float xmax;
   float x;
   float ymin;
   float ymax;
   float y;
   float zmin;
   float zmax;
   float z;
} T_angle;
```

Cette structure contient les valeurs d'angles pour chaque rotation possible (selon x, y et z), ainsi que les valeurs minimales et maximales de celles-ci. Un angle ne pourra jamais dépasser ces valeurs, ce qui permet d'éviter d'avoir des mouvements peu réalistes.

B- Formes de contrôle

1. Gestion des touches multiples

L'utilisation des touches a été modifié par rapport à la structure de base. Au lieu d'agir lorsqu'une touche est activée et répéter l'action jusqu'à ce que la touche soit relâchée,, ce qui provoque un manque de réactivité, nous travaillons simplement sur son état.

Nous avons créé un tableau de booléen contenant une case pour chaque caractère. Lorsque qu'une touche est activée, nous mettons sont booléen correspondant à vrai (1). A l'inverse, quand une touche est relaché, le booléen est remis à faux (0).

Ensuite, dans la fonction principale window_timer, nous réagissons en fonction de l'état des touches qui nous interesse. Ainsi, par exemple, lorsque la touche 'z' est activée mais que la touche 'q' ne l'est pas, nous mettons l'avatar en état de marche. Si les deux touches sont activées, l'avatar sera mis en état de course.

A la fin de ce traitement, nous faisons réagir l'avatar en fonction de son état. Si ce dernier lui demande de marcher, nous activons l'animation de marche et nous faisons avancer l'avatar comme décrit dans la partie III) Animations - A) Mécanisme.

Cela permet de pouvoir tourner en même temps que marcher, ce qui représente un avantage par rapport à la méthode de base pour laquelle il est nécessaire de s'arrêter pour tourner.

2. Mode automatique

Nous avons créée, comme demandé, un mode automatique ou l'avatar réalise des missions.

Ces missions peuvent être diverses et sont appelés aléatoirement. Il peut par exemple vouloir aller à un endroit aléatoire, aller s'asseoir, attendre ou prendre / ranger son épée. Si l'avatar veut aller à un lieu précis, il va d'abord se tourner dans la bonne direction, puis partir en courant. Quand il est assez proche de sa cible, il se met à marcher. Une fois arrivé sur place, il réalise une nouvelle mission aléatoirement.

Si l'avatar veut aller sur la chaise, il ira se positionner devant la chaise, en la contournant s'il est derrière, puis se tournera pour être dos à la chaise avant de s'asseoir.

Lorsqu'il attend, il tapote simplement du pied un certain temps avant de choisir une nouvelle mission et lorsqu'il prend ou range son épée, il appelle l'animation correspondante.

II) Animations

A- Mécanisme

Pour implémenter nos animations, nous avons opté pour une méthode qui nous est propre. Nous avons choisi de miser sur les angles de chaque articulation de l'avatar, et cela pour chacune des étape de chaque animation. Ainsi, toutes les animations que nous avons implémenté ont un tableau d'angles d'articulations pour chaque étape répartis sur le temps.

Chaque animation a un nombre de pas qui lui est propre, et un avancement propre. Ainsi, il est possible de définir un temps plus ou moins important pour passer d'une étape à une autre. L'avancement étant en pourcentage du nombre de pas de l'animation, si on veut accélérer cette dernière il nous suffit de diminuer le nombre de pas.

La lecture d'une animation se fait en calculant l'avancement courant de l'animation pour savoir à quelle étape en est l'animation pour chaque articulation. L'angle suivant (angle cible) est ensuite déterminé pour chaque articulation et la transition d'un angle à un autre se fera linéairement. En effet, comme dit plus haut, nous avons défini une structure T_angle pour chaque articulation. Dans la fonction render_scene(), au moment du placement de tous les membres, nous avons placé des rotations au niveau de toutes les articulation selon les trois dimensions de l'angle comme cela:

```
glRotatef(avatar->angles_art[articulation].x, 1, 0, 0);
glRotatef(avatar->angles_art[articulation].y, 0, 1, 0);
glRotatef(avatar->angles_art[articulation].z, 0, 0, 1);
```

Ainsi, pour passer d'une étape à la suivante, les angles seront incrémentés de manière linéaire pour les trois dimensions d'angle (selon x, y et/ou z) de chaque articulation:

```
void angle_cible(T_angle* angle, float ax, float ay, float az, int nb_steps) {
   if (nb_steps > 0) {
      angle->x += (ax - angle->x) / nb_steps;
      angle->y += (ay - angle->y) / nb_steps;
      angle->z += (az - angle->z) / nb_steps;
   }
}
```

B- Animations créées

Les fonctions demandées ont toutes été implémentées.

Nous avons eu besoin pour cela de connaître la direction de l'avatar (vers où il regarde), et de pouvoir la modifier lorsque l'on veut faire tourner l'avatar.

- La marche:

Plutôt que de déplacer notre avatar à l'aide d'une simple translation avec une valeur qui semble à peu près réaliste, l'avatar se déplace d'une distance issue d'un réel calcul. L'enjambée est calculée, et l'avatar avance en gardant réellement un pied en appui sur le sol. L'avatar ne semble ainsi pas glisser, mais réellement marcher.

Cela est d'autant plus vrai que nous avons implémenté une fonction gardant le bassin à la bonne hauteur par rapport au sol. Il sera en permanence en contact avec le sol, prenant appui d'une manière correspondant à la réalité.

Grâce à la gestion des touches multiples, il est possible de déplacer l'avatar avec aise à l'aide de quatre touches directionnelles.

La vitesse de marche peut être accélérée à l'aide des touches + et -. Il s

- Prendre l'épée/ranger l'épée:

Il est possible de prendre l'épée et de la remettre sur le dos. L'animation a été réalisée à l'aide d'un booléen, qui permet de déclarer l'épée soit dans le dos s'il est faux, soit dans la main s'il est vrai. Arrivé au milieu de l'animation (que ce soit prendre ou ranger l'épée), le booléen change d'état, et ainsi l'épée passe de la main au dos ou du dos à la main.

- La course :

La course diffère de la marche. Ici, la hauteur du bassin n'est pas calculée pour que l'avatar touche le sol en permanence, car dans la réalité, durant la course, il y a un moment où aucun des deux pieds ne touche le sol.

La course de l'avatar est dépendante de l'épée : il court différemment s'il a l'épée en main ou non. Il y a en fait deux animations différentes, qui sont lues en fonction du booléen du paragraphe précédent

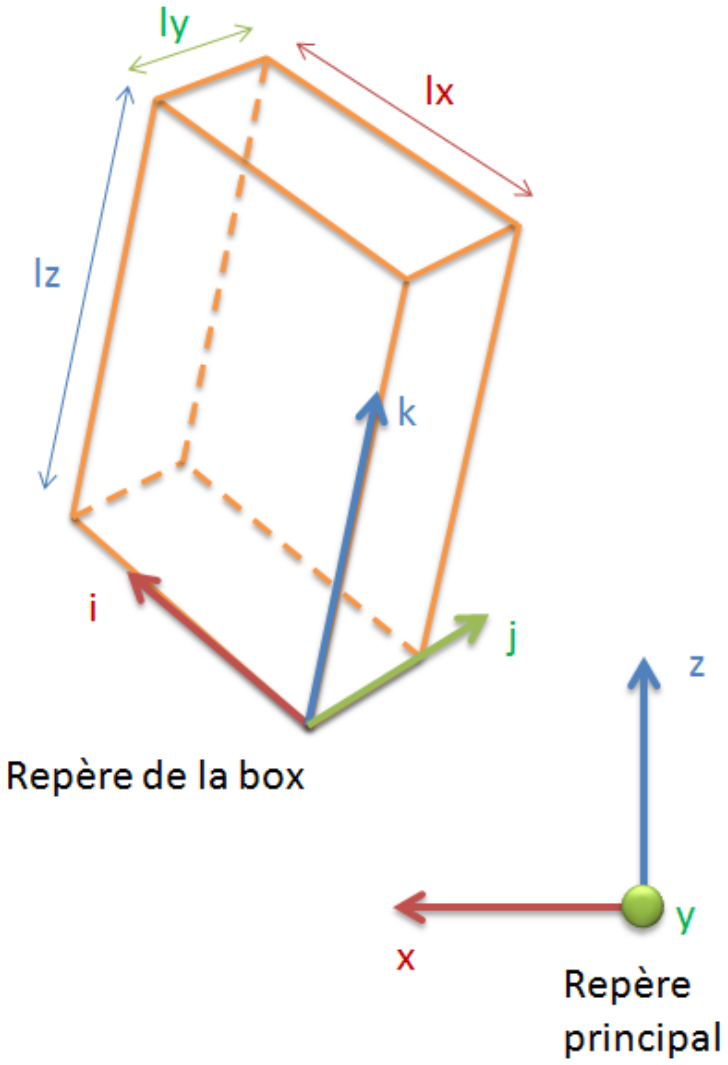
- L'assise :

La position de notre avatar étant calculée à partir du ventre, il nous a fallu changer de repère pour cette animation. En effet, nous voulions conserver les deux pieds au sol, et effectuer la rotation à partir du genou pour que l'avatar s'asseye de façon réaliste.

Nous avons également rajouté la contrainte de ne laisser l'avatar s'asseoir que sur la chaise, et dans le bon sens (grâce à la gestion des collisions)

IV) Gestion des collisions: un moteur physique

Les formes de l'avatar sont complexes, et pour savoir s'il y a un contact entre lui et un autre objet, les calculs sont lourds. Pour contrer cette difficulté, nous avons opté pour la création d'un autre avatar avec des formes basiques, que l'on n'affichera pas, reproduisant chaque partie de l'avatar. Ces formes sont, dans notre cas, des boites représentés par trois vecteurs orthonormés i, j et k, par trois longueurs lx, ly et lz et par la position de la boite (origine de son repère).



Ces boites sont à la base créées sur la base du repère principale, elles subissent ensuite une rotation suivie d'une translation pour se placer au bon endroit. Dans la pratique, ces transformations sont les mêmes que celles que l'on retrouve dans le rendu de la scène, à la différence qu'il est nécessaire de connaître la position des points de la boite pour tester les collisions. L'avatar simplifié aura donc un graphe de scène très similaire à celui de l'avatar.

Ci-dessous, la matrice de translation, avec x, y et z la position du point que l'on souhaite translater, i, j et k les vecteurs formant le repère orthonormé de la boite, et dx, dy, dz le mouvement que l'on souhaite effectuer dans ce repère. Les vecteurs i, j et k doivent avoir une norme de 1.

$$\begin{pmatrix} x' \\ y' \\ z' \end{pmatrix} = \begin{pmatrix} x \\ y \\ z \end{pmatrix} + \begin{pmatrix} i.x & j.x & k.x \\ i.y & j.y & k.y \\ i.z & j.z & k.z \end{pmatrix} * \begin{pmatrix} dx \\ dy \\ dz \end{pmatrix}$$

Ci-dessous, la matrice de rotation, d'angle théta autour de l'axe (a,b,c), passant par l'origine du repère. Ici aussi, le vecteur (a,b,c) doit avoir une norme de 1.

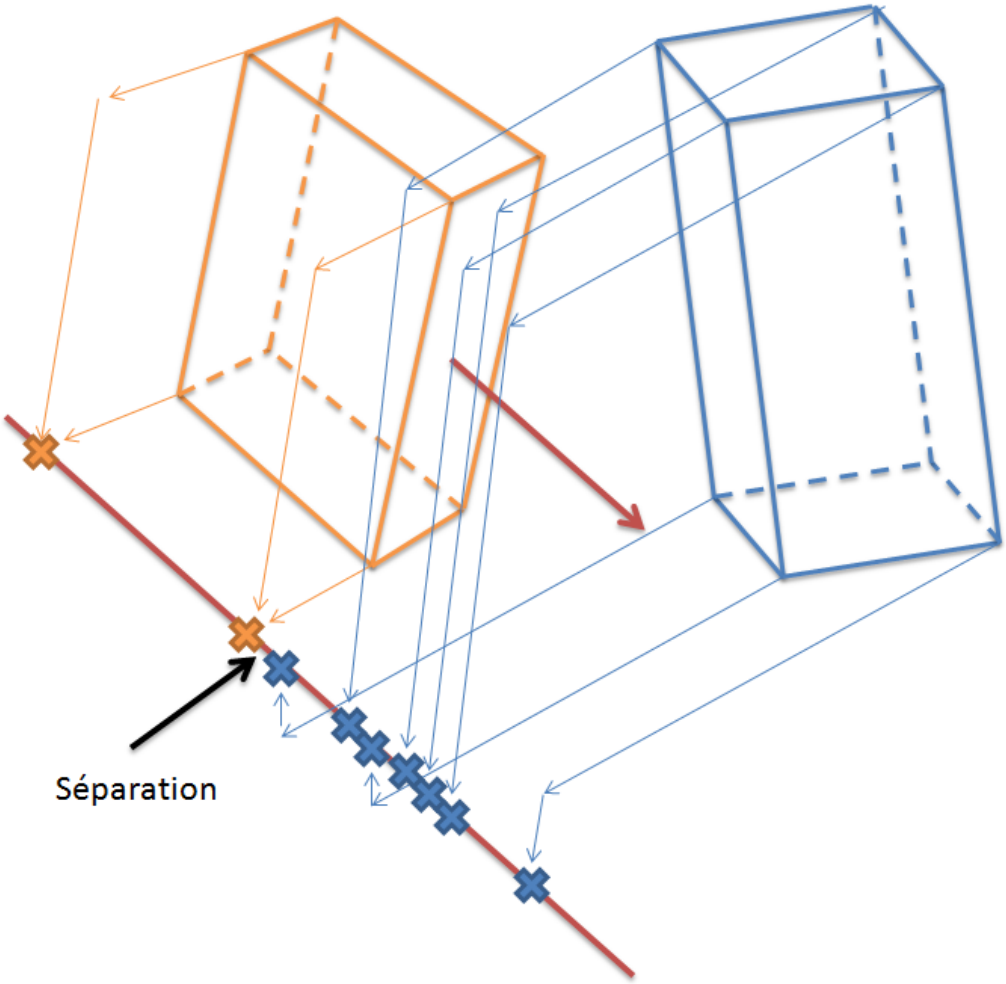
$$\begin{pmatrix} a^2 + (1 - a^2)\,\mathrm{Cos}[\theta] & a\,b\,(1 - \mathrm{Cos}[\theta]) - c\,\mathrm{Sin}[\theta] & a\,c\,(1 - \mathrm{Cos}[\theta]) + b\,\mathrm{Sin}[\theta] \\ a\,b\,(1 - \mathrm{Cos}[\theta]) + c\,\mathrm{Sin}[\theta] & b^2 + (1 - b^2)\,\mathrm{Cos}[\theta] & b\,c\,(1 - \mathrm{Cos}[\theta]) - a\,\mathrm{Sin}[\theta] \\ a\,c\,(1 - \mathrm{Cos}[\theta]) - b\,\mathrm{Sin}[\theta] & b\,c\,(1 - \mathrm{Cos}[\theta]) + a\,\mathrm{Sin}[\theta] & c^2 + (1 - c^2)\,\mathrm{Cos}[\theta] \end{pmatrix}$$

Ces matrices nous permettent de garder un accès sur le repère de la boite, ce qui nous servira à récupérer ses sommets dans le repère principal afin de comparer deux boites dans un repère commun.

Pour tester la collision entre les boites, nous avons décidé d'utiliser le théorème des axes séparateurs.

Deux formes convexes sont disjointes si et seulement s'il existe un plan qui puisse les séparer. Les vecteurs normaux de chaque face définissent un axe. Nous nous focaliserons sur cet axe pour chaque face, car tous les plans séparateurs des deux formes relativement à cette face y seront orthogonaux (car parallèles à la face considérée), s'ils existent.

Pour illustrer ce qu'est un axe séparateur, sur le schéma ci-dessous, nous pouvons en repérer un (en rouge). Il y a en effet une séparation du projeté des 2 boites sur cet axe. Tous les plans orthogonaux à l'axe dont l'intersection avec l'axe est situé entre les projetés bleus et oranges sont donc des plans séparant les 2 boites. Nous pouvons en conclure que ces 2 boites ne sont pas en collision.



Nous récupérons donc pour chacune des faces leur axe orthogonal. Dans notre cas, l'avantage des pavés droits est que ces axes sont simplement les arêtes séparant ses faces.

Nous effectuons ensuite le projeté des points de chacune des boites sur cet axe.

Si l'intervalle sur lequel sont projetés les points de la première boite est distinct de l'intervalle sur lequel sont projetés les points de la seconde boite, alors les deux boites sont séparées et il n'y a pas de collision. On peut donc arrêter l'algorithme ici. En revanche, si les intervalles se croisent, on ne peut pas encore conclure que les boites se chevauchent car il peut exister un autre axe qui sera séparateur.

Il faut donc répéter l'algorithme pour les deux autres axes (il y en a au plus trois par boite), ainsi que pour les axes séparateurs de la seconde boite. De même que précédemment, si le second axe est séparateur, on arrête l'algorithme immédiatement. On agit de la même manière si le troisième axe est séparateur. Si aucun des axes n'est séparateur, c'est que les 2 boites se chevauchent.

V) Commandes de contrôle de l'avatar

Touche	Effet
Z	Marche vers l'avant, ou cours si « a » est pressé
S	Recule en marchant
Q	Tourne vers la gauche
D	Tourne vers la droite
A	Active la course
E	Prend l'épée
R	Range l'épée
I	Active/désactive le mode automatique
+	Augmente la vitesse de marche/course
-	Diminue la vitesse de marche/course
C	Attaque quand l'épée est prise
F	S'assoit sur la chaise
B	Affiche/Cache les boites de collision
*	Désactive les collisions (en cas de blocage)

Conclusion

Au niveau esthétique, nous avons fait de notre mieux pour rendre notre avatar au plus proche du personnage Link : nous lui avons donné une forme proche, et l'avons habillé d'une manière similaire, du bonnet caractéristique aux bottes, en passant par la ceinture et les couleurs. Afin de pousser encore la ressemblance, nous lui avons ajouté une épée qu'il peut saisir et manier quand bon lui semble.

Notre avatar comporte de plus de nombreuses articulations qui permettent des mouvements plus réalistes et fluides. Un système de variables permet de gérer les longueurs caractéristiques de l'avatar sans avoir à changer l'intégralité du code.

Nous avons implémenté un système permettant de gérer l'appui de touches multiples, qui permet un contrôle fluide et immédiat de l'avatar. Un mode automatique a également été ajouté, qu'il est possible d'activer et de désactiver par un simple appui sur une touche. L'avatar se déplacera tout seul à des endroits aléatoires, ou ira s'asseoir.

Pour cet avatar, nous avons fait de notre mieux pour respecter la réalité au plus proche. Les mouvements sont non pas mis à peu près mais calculés, et les animations ont été implémentées dans un souci de réalisme. Les animations ne rendent pas l'avatar rigide et mécanique, mais lui permettent de bouger d'une manière proche de la réalité.

Les collisions sont gérées d'une manière complexe, passant par l'implémentation de l'algorithme des axes séparateurs. Chaque membre de l'avatar possède une boite de collision, et les différents objets de la scène également. L'algorithme permettra de gérer les collisions à partir de ces boites.