IN55 – PROJET

ANIMATION DE PERSONNAGE 3D

Willy KOUETE
Tony DUONG

Introduction

Pour notre premier projet en synthèse d'image, nous avons choisi l'animation de personnage 3D. En effet, étant intéressé par la 3D et le films d'animation en 3D tels que ceux produits par Pixar, Disney ou encore Dreamworks, ce premier projet est une opportunité pour nous de nous plonger dans ce monde et de découvrir les premiers aspects de l'animation des personnages tel qu'on le voit dans ces films d'animation.

Dans notre projet, nous nous sommes concentrés plus particulièrement à la partie « animation » dans OpenGL au détriment de la partie « modélisation » auquel nous avons jugé que c'était plutôt le travail des designers.

Le but de ce projet est d'animer un personnage en 3D en utilisant OpenGL. Notre challenge a été donc de réaliser les mouvements de notre personnage de la manière la plus réaliste possible.

Dans ce rapport, nous vous expliquerons plus en détail les différents choix que nous avons faits afin de mener à bien ce projet.

SOMMAIRE

A. Les technologies et ressources utilisées

Le projet a été réalisé sous l'IDE Qt Creator en langage C avec la librairie OpenGL et les shaders GLSL.

Un choix à faire a été le format d'export du fichier de notre modèle 3D. MD5 et Collada sont deux formats très populaires pour l'animation de personnages.

L'avantage du Collada est qu'il contient beaucoup d'informations. En effet, le fichier Collada contient la scène entière avec les modèles, les illuminations, les matériaux etc…

MD5 est aussi un format qui correspond à nos besoins bien que contenant tout juste l'information nécessaire pour le rendu et l'animation du personnage.

Nous avons finalement choisi le format Collada car nous avons voulu apprendre à utiliser un format complet qui nous permettrait le rendu de scènes entières.

Pour lire et récupérer les informations de ces documents Collada qui sont des fichiers XML, la librairie tinyxml a été utilisée.

Du côté du modèle 3D utilisé, le modèle disponible à l'adresse suivante a été utilisé pour notre projet : http://www.wazim.com/Downloads.htm

B. Collada

Collada (Collaborative Design Activity) a pour but d'établir un format de fichier d'échange pour les applications 3D interactives. Les documents Collada sont des fichiers XML, habituellement identifiés par leur extension *.dae*.

Notre première tâche a été de créer un parseur de fichier Collada. Nous aurions pu utiliser des librairies qui peuvent importer des fichiers Collada tel que Assimp. Mais nous avions préféré écrire notre propre parseur pour pouvoir connaitre en détail tous les éléments qui composent un fichier Collada et leur utilisation par rapport à notre projet.

1. Diagramme de classes

Ecrire ce parseur a consisté à parcourir le fichier XML et à « traduire » celui-ci en C++. En d'autres mots, les nœuds dans le fichier XML seront considérés comme des classes et les nœuds fils seront eux-mêmes considérés comme des éléments de la classe associée au nœud parent.

```
<library_animations>
  <animation id="astroBoy_newSkeleton_root-transform">
    <source id="astroBoy_newSkeleton_root-transform_astroBoy_newSkeleton_root_transform_0__0_-input">
```

Figure 1 - Collada structure

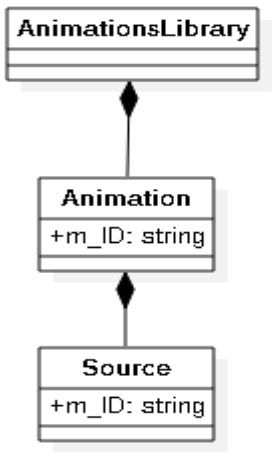
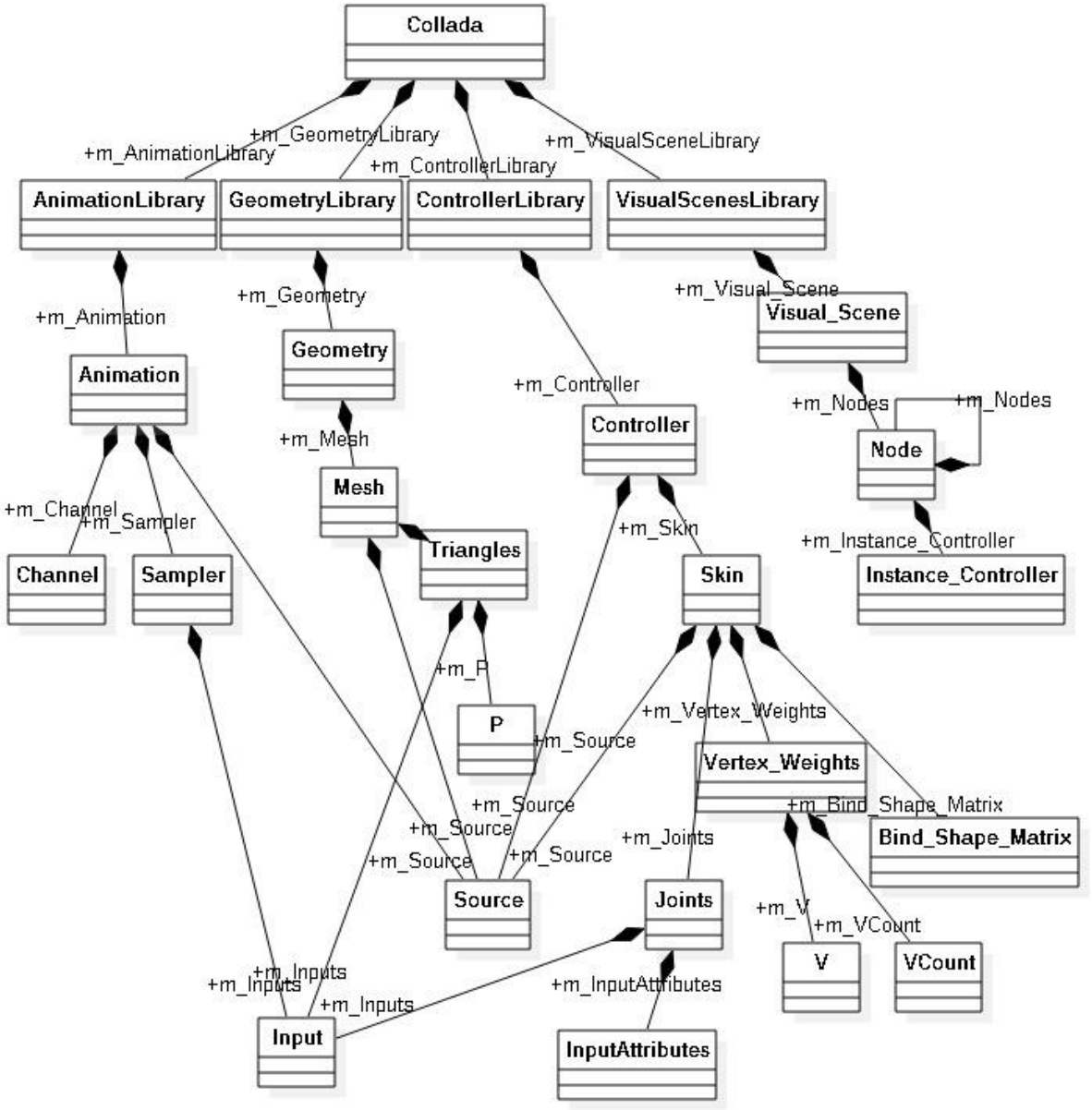


Figure 2 - l'équivalent en C++

Le diagramme de classe



L’intégralité du diagramme n’est pas représentée ici. Nous avons seulement gardé ceux que l’on a utilisé pour notre projet.

Je n’expliquerai pas en détail le diagramme (cela prendrait beaucoup de temps et n’a pas d’importance pour la bonne compréhension de notre travail). Il faut savoir que l’AnimationLibrary est utilisé pour les données d’animation comme les données temporelles ou les données de position des joints (ce concept sera expliqué ultérieurement dans le rapport). Ensuite, la GeometryLibrary est utilisé

pour récupérer les coordonées des vertices ainsi que des textures. Enfin, les ControllerLibrary ainsi que la VisualSceneLibrary eux permettent de récuperer les transformations des joints afin de modifier la pose du personnage.

2. Utilisations des données récupérées

Une fois les informations du fichier Collada extraites, il faut les traiter.

Vertex skinning

Le vertex skinning est l'étape d'animation des vertices d'un modèle 3D en les contrôlant via un squelette. Si le squelette bouge, les vertices (donc la peau) bougeront aussi.

Les données qui nous ont été utiles pour notre projet sont :

- Les coordonnées (x, y, z) des vertices de la position initiale aussi appelée *bind pose*

  La bind pose des *bones* qui constituent le squelette est simplement la position de départ de ceux-ci. Lors de la modélisation du modèle, c'est la pose sur lequel le modeleur ajoute le squelette (ou *skeleton*).

- Les matrices de transformation des joints ou *joint matrix*

  Ces *joint matrix* servent à modifier la position et l'orientation des joints du squelette. Par la même occasion, du fait que les vertices peuvent être influencés par un ou plusieurs joints, ces derniers verront également leur position modifiée. De plus, il existe une hiérarchie en arborescence entre les joints. Le squelette possède nécessaire un joint racine (*root joint*) et celui-ci a plusieurs joints fils qui sont eux-mêmes influencés par leur joint parent.

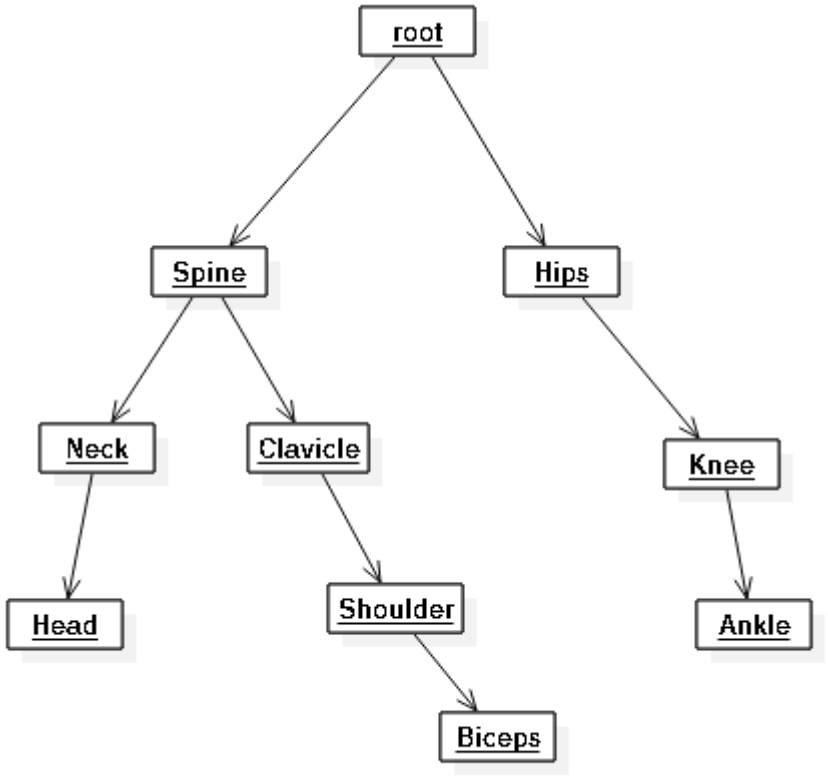


Figure 3 - Hiérarchie des joints

Dans le cas de la figure 3, si le joint *Spine* est déplacé alors tous ces joints enfants mais aussi les joints enfants de ses enfants, le *Neck, Head, Clavicle*… seront aussi déplacés. Dans le cas contraire si le *Head* est déplacé, il n'influera pas du tout sur les joints du squelette.

Pour produire cet effet de dépendance, on multiplie la matrice de joint courante ou locale à celle de ses joints enfants. Et le résultat sera sauvegardé en tant que matrice de transformation globale dans chacun des fils.

- La *inverse bind matrix*

  Cette matrice est nécessaire car elle permet de translater le repère global à un repère local où le joint correspondant à cette *inverse bind matrix* est situé au (0,0,0) de ce repère local.

- Les poids associés à chaque couple (vertice, joint)

  Chaque vertice peuvent, comme dit auparavant, être influencé par un ou plusieurs joints. Néanmoins, ces joints n'influent pas tous de la même

manière sur un vertex. En effet, le joint le plus proche du vertex aura une plus grande influence sur ce dernier. C'est pourquoi les poids ou *weights* sont associés à chaque couple (vertex, joint).

Note : la somme des poids sur un vertex doit être égale à 1.

Après avoir réuni toutes ces données, on applique l'équation suivante :

$$nouvPosVertex = \sum_{i=0}^{n} (v * IBM_i * JM_i) * w_i$$

Texture mapping

Dans le fichier Collada, il a aussi été possible de récupérer les coordonnées (x, y, z) des textures associées aux vertices. Pour effectuer ce mapping, nous avons utilisé les Shaders. Voici la texture utilisée pour notre personnage.

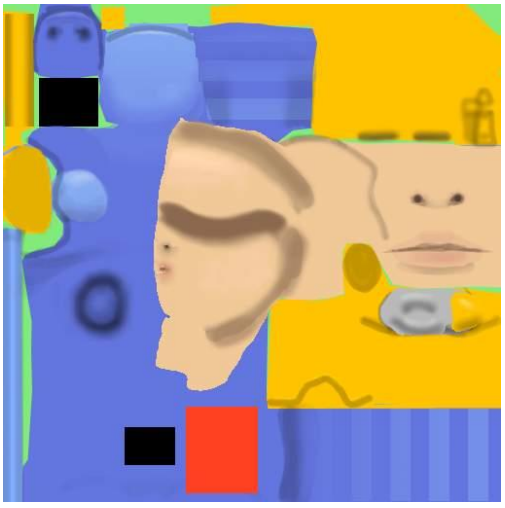


Figure 4 - Texture

Vertex and Fragment Shaders

```
#version 140

uniform mat4 MVP;

in vec3 position;
in vec2 texCoord;
in vec3 color;

out vec3 fColor;
out vec2 fTexCoord;

void main()
{
  gl_Position = MVP * vec4( position, 1.0f );

  fColor = color;
  fTexCoord = texCoord;
}
```

```
#version 140

in vec3 fColor;
in vec2 fTexCoord;
uniform sampler2D texId;

out vec4 fragColor;

void main()
{
    vec4 texColor = texture2D(texId, fTexCoord);
    fragColor = texColor;
}
```

3. Animation

Après avoir attaché le squelette au modèle grâce à l'équation du skinning précédemment évoquée, tous les vertices du modèle sont influencés par ce squelette. Ainsi la position de chaque vertice sera déterminée par un état des valeurs de certaines matrices représentant les joints. Pour obtenir une séquence représentant une animation, il est donc plus aisé de calculer les séquences de transformation des joints, puis d'appliquer chaque transformation obtenue aux vertices influencés.

Le modèle utilisé offre une séquence de joints par défaut. Cette séquence représente les différentes positions stratégiques de l'animation. Il est donc compréhensible que s'en tenir la séquence offerte par le modèle rendra l'animation moins fluide et donc pas réaliste.

Une solution serait de procéder à une interpolation afin d'obtenir les différentes positions intermédiaires entre deux positions par défaut. La mise en place de cette solution consiste à calculer les différents joints intermédiaires.

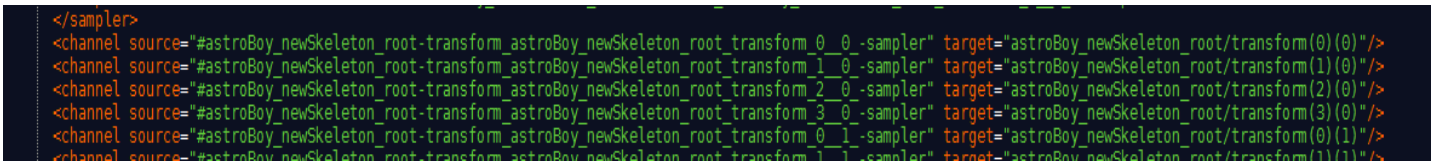
Il s'agira de calculer par interpolation linéaire les éléments de la matrice de joint intermédiaire. La matrice ainsi obtenu servira de *Joint Matrix* et pourra ensuite être utilisée via l'équation de skinning pour déduire les positions futures des vertices.

La procédure d'animation du squelette avec Collada

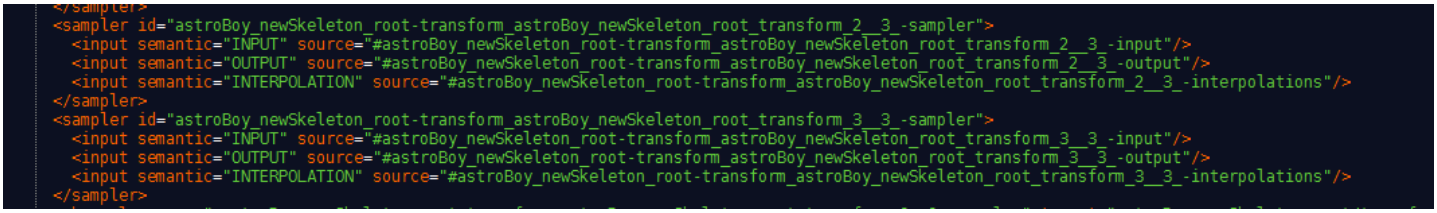
➔ Les données d'animation sont récupérées des nœuds du nœud ***AnimationLibrary***.

Pour chaque animation :

- Récupération de tous les identificateurs des sources des données temporelles et matricielles ainsi que l'identificateur du joint qui subit les transformations, et les coordonnées matricielles de la transformation en cours. Sous format Collada, ces données sont stockées dans les nœuds ***channel***.



- Recherche des sources de données à partir des identificateurs précédemment récupérés : la recherche des identificateurs s'effectue au niveau des nœuds ***sampler,*** et chaque identificateur référence un nœud ***source*** dans le format Collada. Le nœud ***sampler*** contient principalement 3 nœuds dont 2 nous intéressent : le nœud à sémantique ***INPUT*** référençant les sources de données temporelles, et le nœud de sémantique ***OUTPUT*** référençant les sources de données matricielles.



- Pour chaque source de donnée trouvée, récupération des données temporelles : en format Collada, cette source est indiquée par le nœud fils de sémantique ***INPUT*** dans le nœud ***sampler*** et contient pour chaque position du squelette le temps associé.

- Les données matricielles sont également récupérées à partir des sources fournies par le nœud fils de sémantique ***OUTPUT*** du nœud ***sampler***
-

```
<source id="astroBoy_newSkeleton_root-transform_astroBoy_newSkeleton_root_transform_3__2_-input">
  <float_array id="astroBoy_newSkeleton_root-transform_astroBoy_newSkeleton_root_transform_3__2_-input-array" count="36">0 0.033333 0.066666 0.1 0.133333 0.166667 0.2 0.23
  <technique_common>
    <accessor source="#astroBoy_newSkeleton_root-transform_astroBoy_newSkeleton_root_transform_3__2_-input-array" count="36" stride="1">
      <param name="TIME" type="float"/>
    </accessor>
  </technique_common>
  <technique profile="MAYA">
    <pre_infinity>CONSTANT</pre_infinity>
    <post_infinity>CONSTANT</post_infinity>
  </technique>
</source>
<source id="astroBoy_newSkeleton_root-transform_astroBoy_newSkeleton_root_transform_3__2_-output">
  <float_array id="astroBoy_newSkeleton_root-transform_astroBoy_newSkeleton_root_transform_3__2_-output-array" count="36">2.6942 2.6942 2.68355 2.67173 2.67849 2.79133 2.9
  <technique_common>
    <accessor source="#astroBoy_newSkeleton_root-transform_astroBoy_newSkeleton_root_transform_3__2_-output-array" count="36" stride="1">
      <param type="float"/>
    </accessor>
  </technique_common>
</source>
```

➔ Les données récupérées sont ensuite appliquées aux différentes matrices de joint grâce aux coordonnées matricielles récupérées précédemment du nœud ***channel***

Dès lors, il revient juste à recalculer les matrices de transformation globale des joints du squelette en prenant en compte les matrices issues du remplacement des joints par les données matricielles d'animation lues précédemment.

Le rendu de ce nouveau squelette fournira la position suivante du squelette, et donc des différents vertices, puisque ces vertices sont influencés par les joints du squelette.

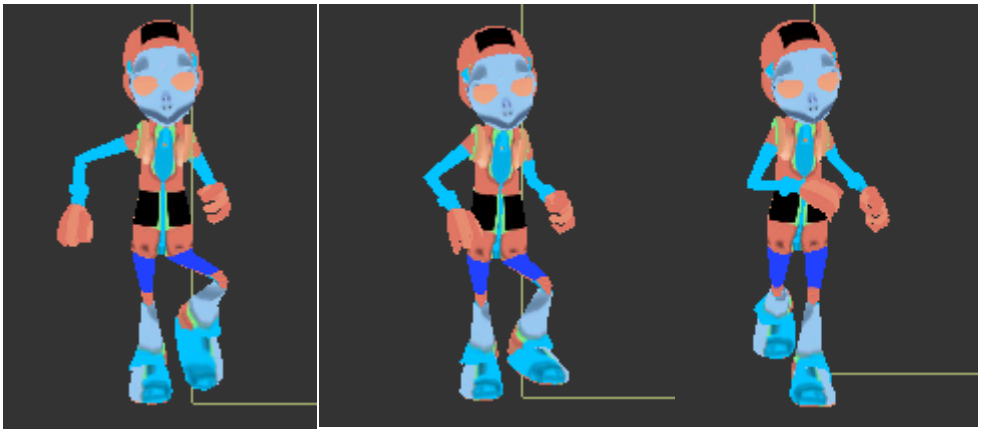


Figure 5 - Exemples de keyframes

Cependant, s'en tenir seulement aux positions du squelette telles que fournies par la modélisation pour effectuer l'animation ne la rendra pas très réaliste, car il y aurait des sauts de frame. Alors la solution est de procéder à une interpolation des joints.

Cette opération aura pour but d'estimer, en un temps *t*, l'état de valeurs matricielles situées entre deux états bien connus, offert par le modèle. En d'autres termes, il s'agira de déterminer les différents frames manquants entre deux positions du squelette, afin de garantir un réaliste optimal de l'animation.

Plusieurs méthodes d'interpolation existent, allant de l'interpolation linéaire à d'autres plus complexes.

Pourquoi l'interpolation linéaire ?

L'interpolation linéaire est la méthode d'interpolation qui se présente à nous comme la plus simple. Son application permet d'obtenir linéairement les différentes positions situées entre deux positions connues à 2 instants distincts.

Cependant ce type d'interpolation présente vite des limites lorsqu'il s'agit d'obtenir également les différentes orientations à ces instants. Elle est donc généralement couplée avec l'interpolation SLERP pour l'interpolation des orientations.

Cependant dans notre cas, les matrices de joint contiennent déjà toutes les données d'orientation du personnage. L'interpolation linéaire se présente donc comme le choix le plus évident.

Rappel d'interpolation linéaire

Soit $P_i \; et \; P_{i+1}$ les positions aux temps respectifs $t_i \; et \; t_{i+1}$. Alors la position P obtenue par interpolation linéaire entre $P_i \; et \; P_{i+1}$ au temps t tel que $t_i \leq t < \; t_{i+1}$ est données par la relation :

$$P = (1-u)P_i + uP_{i+1}\ avec\ u = \frac{t-t_i}{t_{i+1-t_i}}$$

Application de l'interpolation linéaire au squelette

Dans notre cas, les valeurs de *P* sont les différentes valeurs des différents éléments de la matrice de squelette. Il s'agira donc de calculer à chaque pas de temps les valeurs de la matrice obtenue par interpolation de deux matrices consécutives.

Pour procéder à l'interpolation linéaire, il faut d'abord déterminer les instants $t_i\ et\ t_{i+1}$ qui encadrent directement l'instant auquel on souhaite obtenir le frame, et calcule le facteur d'interpolation *u* comme donné ci-dessus.

Ensuite, retrouver les matrices correspondantes à ces instants. Puis, pour chaque élément *i* des matrices obtenues, calculer l'élément interpolé en application l'équation d'interpolation ci-dessus.

La matrice obtenue sera celle représentant le joint en question à l'instant *t*. Cette matrice sera donc utilisée en remplacement de la matrice de joint initiale, et le calcul des transformations globales et leur application aux vertices fourniront la position manquante de notre personnage à l'instant souhaitée.

Le personnage animé retrouvera ainsi les frames manquants afin de rendre son mouvement plus fluide. Plus le pas de temps sera petit, moins les sauts de frame seront fréquents, et le personnage sera plus réaliste côté mouvement, moyennant bien sur le coût de calcul du processeur.

C. Résultats

A partir des vertices de base (où la transformation n'a pas été encore appliqué), nous avons pu faire le rendu du modèle en pose normale (la pose que le designer a modélisé au départ).

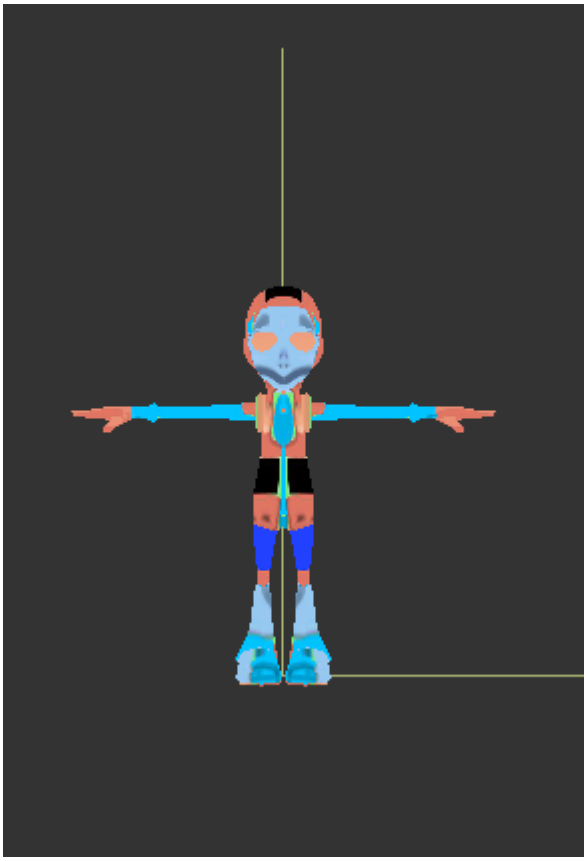


Figure 6- pose de base

Ensuite, après avoir appliqué toutes les matrices de transformations qui permettent de changer la position et l'orientation des joints (et donc par la même occasion les vertices), nous avons pu faire le rendu de la première keyframe d'animation.

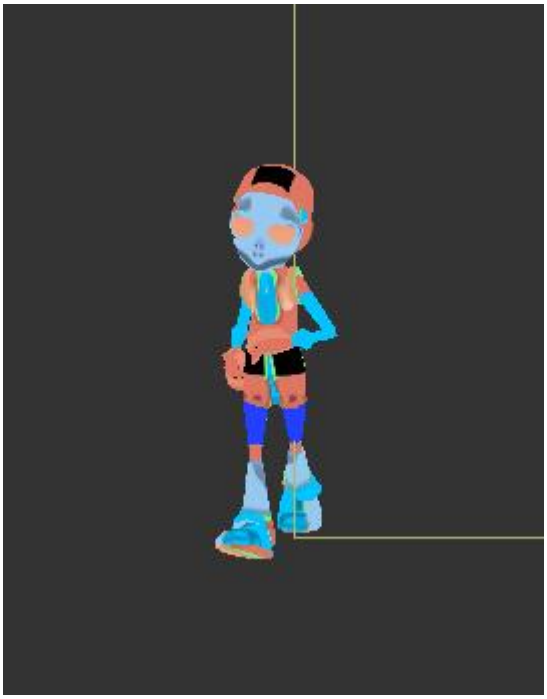


Figure 7 - première keyframe

Enfin, en récupérant toutes les données d'animations dans le fichier permettant d'afficher le personnage à différentes keyframes et en utilisant la technique d'interpolation, nous avons pu animer le personnage.

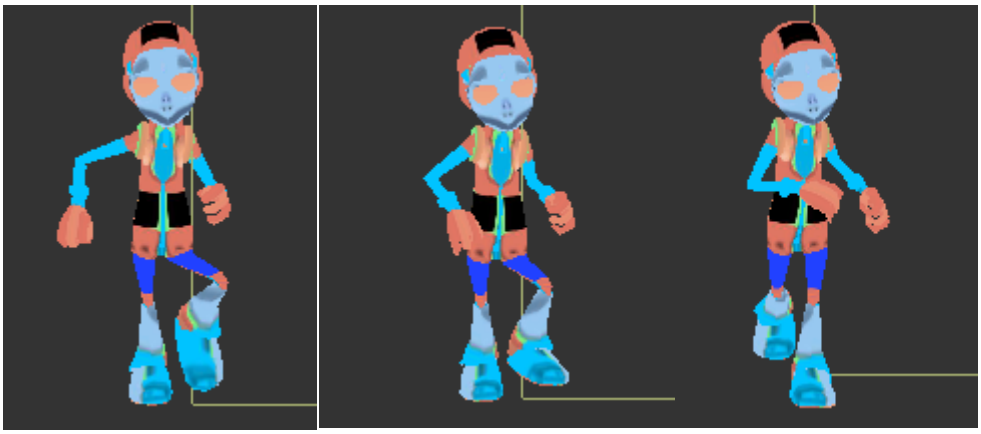


Figure 8 - Animation par interpolation des keyframes

Difficultés rencontrées et améliorations possibles

Pendant notre projet, nous avons rencontrés plusieurs difficultés.

Tout d'abord, bien que l'écriture du parseur Collada n'a pas été très difficile, l'utilisation des données extraites l'a été. En effet, nous avions passé beaucoup de temps à faire le rendu de la première keyframe d'animation car nous avions rencontré un problème concernant le calcul des matrices de transformation globale. Mais après multitudes de tests et de recherches sur Internet, nous avions fini par trouver ce qui n'allait pas.

Puis les couleurs affichées sur le personnage ne sont pas ce que l'on attendait. Elles ne correspondent pas à ceux qui sont sur la texture (*Figure 4*). Nous pensons que la cause provient de la non-intégration de l'illumination dans notre scène.

Enfin, en ce qui concerne la caméra, nous n'avions pas eu le temps d'implémenter une caméra libre faute de temps. Nous avions utilisé celui du TP1qui nous permet quand même d'observer le personnage en animation sous différents angles.

Conclusion

Nous avons réussi à animer un personnage en utilisant la méthode du *Skeletal Animation.* Ce projet nous a beaucoup appris de choses sur l'animation de personnage ou plus généralement de modèles 3D. En effet, les concepts étudiés dans nos recherches peuvent être appliqués à toute sorte de modèles 3D.

Même si tout n'a pas été couvert dans notre projet comme l'illumination ou les matériaux, nous avons pu faire le rendu d'un personnage, l'animer et lui appliquer une texture. L'écriture du parseur Collada nous a vraiment permis de comprendre en détail la structure du fichier et la manière d'utiliser les données extraites.

Sources

Collada Specifications V1.4

https://www.khronos.org/files/collada_spec_1_4.pdf