Année universitaire 2021-2022

Majeure IMI – Partie 4 – 5ETI

Animation et simulation

Compte Rendu TP Skinning

NICERON Simon/LOUBEYRE Emma/DEDDAH Tahya

I. Introduction

Dans l'histoire de l'animation, de nombreuses techniques ont été mises en place pour donner vie à des personnages. Ces techniques de base ont été développés, puis améliorés jusqu'à aujourd'hui où elles sont d'un grand intérêt pour l'industrie de l'audio-visuel. L'une de ces techniques est la méthode de déformation par Skinning qui lie une surface à un squelette d'animation à l'aide d'un ensemble de poids d'influences. Celle-ci est très utilisée dans le domaine du cinéma et du jeu vidéo pour déformer des personnages animés car elle permet un contrôle intuitif et une évaluation rapide. Le skinning permet donc d'animer des personnages 3D. Pour ce faire, cela consiste à donner à un personnage tous les éléments nécessaires à son animation, et notamment à lui associer un squelette. On peut comparer cette structure de squelette à la nôtre car elle est composée d'os et de point de jonction. Lorsqu'on lui associe un squelette, il faut également lui associer finement des sommets à tel ou tel Bone, avec en plus un poids associé car les poids servent à attirer plus ou moins chaque sommet vers l'os qui se déplace. Et donc un sommet peut se déplacer en fonction de plusieurs os s'il a plusieurs poids qui lui sont affectés. Ces poids de skinning sont fixés manuellement pour obtenir une déformation très réaliste pour obtenir un résultat optimisé, ce qui peut être contraignant.
Les avantages de cette méthode sont qu'il est facile de construire un squelette, facile de l'animer et intuitif de manipuler un personnage articulé. Le seul désavantage est qu'on observe des discontinuités possibles.
Nous allons donc appliquer cette méthode Skinning pour déformer un cylindre puis par la suite un personnage.

II. Objectifs

L'objectif de ce TP est de coder une déformation d'objet 3D par la méthode skinning qui consiste à déformer une surface en suivant un squelette articulé guidant différentes parties de l'objet. Pour ce faire, nous générons la déformation d'un objet simple de type cylindre formé de deux os. Pour ce faire, nous allons dans un premier temps créer un cylindre simple, son squelette animé en utilisant trois joints distincts associés à un repère local et global, puis nous allons créer une déformation de la surface avec attribution des poids de skinning et d'interpolations. Enfin, dans un second temps, nous appliquerons la déformation sur un personnage complet à partir d'un fichier.

III. Prise en main de l'environnement

Pour ce TP, nous travaillons sur un IDE qui sera Visual Studio code. Nous utiliserons Cmake pour dire à notre compilateur comment compiler le code. La bibliothèque glew est disponible pour gérer les extensions d'openGL et Freeglut aussi pour manipuler les fenêtres contenant les projets OpenGL.
Certaines parties du programme sont déjà existantes et d'autres sont à faire pour réaliser la déformation d'objet 3D par la méthode skinning. Nous retrouvons les répertoires ayant déjà servis dans d'autres projets tels que lib contenant l'ensemble des bibliothèques, image puis d'autres comme local contenant la scène et l'interface où nous travaillerons, external/perlin contenant un bruit simple et skinning le répertoire qui contient les classes permettant la gestion d'un squelette d'animation, des poids de skinning et d'un maillage déformé par skinning.
En exécutant le programme, nous observons sur l'écran seulement un plan. Nous nous intéressons alors à la structure de la classe *scene* pour comprendre son fonctionnement. Dans cette classe, nous allons construire le maillage ajouté à un *mesh* et établir les relations entre les différents indices pour créer nos triangles et donc créer notre objet. Nous allons également appeler les fonctions utilisées dans le *load_scene()* et les dessiner dans le *draw_scene()* pour visualiser par la suite notre objet.

IV. Skinning d'un cylindre

Dans cette partie, nous allons créer tout d'abord un cylindre géométrique sous forme de maillage. Ensuite, nous créerons un squelette géométrique qui lui est associé puis animerons ce squelette. Enfin, nous attacherons des poids de skinning aux sommets du cylindre puis déformerons cette surface (flexion du cylindre) en fonction de l'animation du squelette.

3.1 Surface géométrique

Pour créer un cylindre, c'est-à-dire créer le maillage du cylindre, nous allons utiliser la classe *mesh_cylinder* de type *mesh_skinned*. Nous allons pour cela, utiliser plusieurs paramètres définissant le cylindre et les utiliser comme argument dans la fonction :

- N_u le nombre de cercle, c'est-à-dire le nombre « d'étages » composant le cylindre
- N_v le nombre de points sur le cercle, c'est-à-dire le nombre de points composants la circonférence du cylindre
- L la longueur totale des cercles, c'est-à-dire la hauteur du cylindre

- R le rayon

Avec les points N_v, nous pouvons former chaque cercle composant « l'étage » du cylindre (nous avons 3 cercles dans notre cas à former) puis avec N_u, nous pouvons relier les cercles entre eux afin de former un cylindre. Chaque point N_v est défini comme: $\frac{2*\Pi}{N_u} * k_u$.
Nous orientons notre cylindre suivant l'axe z. Pour dessiner le maillage, il suffit de tracer plusieurs triangles qui composeront l'ensemble du maillage, c'est-à-dire que nous allons ajouter chaque point du maillage de notre cylindre avec l'appel de la fonction *add_vertex()*, en prenant en compte sa composante selon z puis relier ces points afin de former notre cylindre composé d'un ensemble de triangles grâce à la fonction *add_triangle_index(vertex1, vertex2, vertex3)* définissant la connectivité entre les triangles.

Dans la fonction *load_scene()*, on appelle notre fonction de création de cylindre afin de compléter la variable *mesh_cylinder* qui permettra de charger le maillage de notre cylindre. On transmet alors toutes les données composants notre cylindre (textures, couleurs, normales) sur le GPU en faisant appel à la fonction *fill_vbo()* puis on affiche notre cylindre grâce à la fonction *mesh_cylinder_opengl.draw()* pour pouvoir visualiser notre cylindre dans la scène avec le plan.

Nous obtenons les résultats suivants :

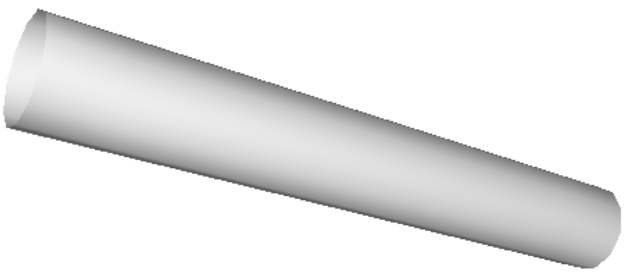

Figure 1: Création du maillage du cylindre

3.2 Squelette du cylindre

Nous avons pu jusqu'ici créer un objet de type cylindre pour lequel nous allons implémenter son squelette afin d'effectuer par la suite une déformation.

3.2.1 Position de repos

Pour créer un squelette, il faut tout d'abord créer des joints. On crée des joints de manière hiérarchique car si un joint est parenté sous un autre joint alors ces deux joints seront connectés par un os (bone). Nous pouvons représenter les joints comme étant les articulations de nos personnages qui sont connectés par des os. L'intérêt est alors de bien placer et orienter ces joints à l'intérieur d'un modèle 3D pour permettre l'articulation souhaitée.

Dans le cas de notre TP, nous allons créer le squelette du cylindre correspondant à la géométrie actuelle de la surface que l'on désigne par position de repos, ou encore bind pose car c'est dans cette pose que nous allons attribuer les poids de skinning par la suite.
Pour ce faire, nous allons considérer que le cylindre est formé d'une hiérarchie simple de deux os mis bout à bout et de 3 positions (joints) situés aux extrémités du cylindre permettant de définir ces deux os. Chaque position j_0, j_1 et j_2 est associée à un repère propre défini localement.

Ce squelette va être défini à l'aide de deux structures.

- Une première structure *skeleton_parent_id* va stocker la connectivité de la hiérarchie et plus précisément l'indice du joint parent au joint étudié. Nous avons donc : parent(0)=-1, parent(1)=0, et parent(2)=1. Le premier joint n'a pas de parent, c'est pourquoi on lui attribue la valeur -1.

- Une seconde structure géométrique *sk_cylinder_parent_id* qui va stocker chaque repère de joint à l'indice correspondant. Chaque repère est défini de manière locale, c'est-à-dire que le repère du joint 1 est défini par rapport au repère 0, et le repère du joint 2 est défini par rapport au repère du joint 1.

On doit définir les positions du squelette en fonction des translations et rotations, et donc les définir en fonction des parents. Nous n'avons pas de rotation donc nous mettons les quaternions à 0. On attribue l'identifiant −1 au joint j_0 , 0 au joint j_1 et 1 au joint j_2.
L'expression du repère du joint j_1 par rapport au repère du joint j_0 est alors :

R_1= position j_0 + (0 0 $\frac{L}{2}$)

L'expression du repère du joint j_2 par rapport au repère du joint j0 est alors :

R_2 = position j_1 + (0 0 $\frac{L}{2}$)

Les positions des joints par rapport à leur parent sont alors stockées suivant :

```
skeleton_joint j0(vec3 (0,0,0), quaternion(0,0,0,1));
skeleton_joint j1(vec3 (0,0,L/2), quaternion(0,0,0,1));
skeleton_joint j2(vec3 (0,0,L/2),quaternion(0,0,0,1));
```

Après avoir décrit localement nos positions, nous avons besoin de leurs coordonnées dans le repère global pour visualiser le squelette. Nous devons alors convertir ces coordonnées locales en coordonnées globales. Nous avons la relation suivante pour cette conversion :

$$G_j = G_{p(j)} L_j$$

G_j est la matrice constituée des coordonnées globales du repère lié au joint j, $G_{p(j)}$ la matrice des coordonnées globales du parent du joint j, et L_j la matrice des coordonnées locales de j.

Pour exprimer les transformations isométriques, c'est-à-dire exprimer les rotations et translations, nous utilisons la matrice M formée d'un quaternion q représentant la rotation par rapport au repère parent, et d'un vecteur t représentant la translation par rapport au repère parent également :

$$M = \begin{pmatrix} q & t \\ 0 & 1 \end{pmatrix}$$

L'expression du quaternion q_j et de la translation t_j exprimés dans le repère global en fonction des quaternions et translations du repère j exprimés localement et des quaternions et translations du repère p(j) exprimés globalement est :

$q_j = q_{p(j)}$ *q

$t_j = q_{p(j)}$ *t + $t_{p(j)}$

Avec cette expression, nous pouvons exprimer les repères associés aux joints du squelette dans le repère global en fonction d'un squelette dont les repères des joints sont exprimés de manière locale.

Nous cherchons à afficher le squelette, pour ce faire nous devons afficher les segments joignant deux joints consécutifs dans la hiérarchie. Soit le fonction *extract_bones* qui vient générer un vecteur stockant ces os sous la forme de paire de points 3D :

```
vec3 const pos0=skeleton[k].position ;
vec3 const pos1=skeleton[parent].position ;
positions.push_back(pos1) ;
positions.push_back(pos0) ;
```

Il ne reste plus qu'à afficher le squelette dans le fonction *draw_skeleton()* en l'appliquant sur le vecteur contenant la position des os.
Nous obtenons le résultat suivant:

Figure 2: Création du squelette du cylindre

3.2.2 Animation du squelette

Nous souhaitons animer le squelette après avoir défini les positions respectives de chaque joint dans le repère global et en fonction des joints parents, c'est-à-dire définir l'orientation et la position des joints à chaque instant considéré.
On choisit 4 poses courantes du squelette (t=0 jusqu'à t=3) correspondants respectivement à des angles de 0°, 30°, 60° et 90°.

Soit la figure ci-dessous montrant l'animation du squelette du cylindre pour 4 poses clés :

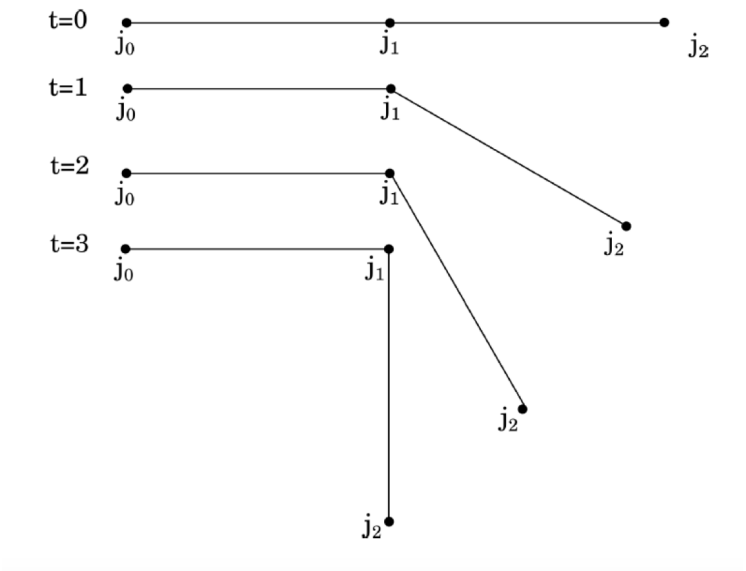

Figure 3: Animation du squelette du cylindre pour 4 poses clés

D'après la figure, seule la position et l'orientation du joint j_1 va être impactée.

On crée une fonction *remplir_animation* qui va stocker les 4 poses courantes du squelette, regroupant toutes les géométries des squelettes selon l'angle donné. Pour obtenir les positions suivant la figure, nous devons appliquer la rotation au joint 1, c'est-à-dire appliquer au quaternion du joint 1 la fonction *set_axis_angle()* permettant de définir l'axe et l'angle de la rotation à effectuer. L'angle variera de 30 ° pour chaque nouvelle géométrie des squelettes. Nous devons créer une boucle *for* dans *scene.cpp* pour mettre en place une transition automatique visualisant l'animation du squelette en boucle. Une fois cette boucle mise en place, la variable indiquant la pose courante du squelette s'incrémentera toute les x millisecondes. Nous choisissons que cette transition entre les différentes poses se fasse automatiquement après 1s. Nous réinitialisons le temps à 0 à chaque changement de pose.

Nous devons faire attention à l'orientation du joint car un joint mal orienté peut ainsi entraîner des rotations sur plusieurs axes, ce qui peut rendre le travail de l'animateur très difficile. Les valeurs de rotation de chaque joint doivent être cohérentes par rapport au mouvement que l'on souhaite obtenir au final. Mieux vaut alors préserver des valeurs positives dans le sens où l'articulation se fait.

3.3 Déformation de la surface
3.3.1 Attribution des poids de skinning

Nous souhaitons déformer la surface de manière à ce qu'elle suive le mouvement de notre squelette que nous venons de définir suivant plusieurs poses courantes. Nous devons donc attacher la surface à son squelette afin de la déformer suivant le mouvement des os.
Pour réaliser cette déformation, nous devons tout d'abord attribuer les poids de skinning entre chaque sommet de la surface et les repères du squelette. Les poids de skinning définissent donc l'attachement d'un vertex du maillage aux joints du squelette.
Nous considérons que la première moitié du cylindre suit rigidement le premier os composé des joints $j_0$et j_1. Dans cette première moitié, les sommets auront un poids de 1 par rapport à $j_0$et un poids de 0 par rapport à j_1. De la même manière, nous allons attribuer ces poids pour la deuxième moitié du cylindre, les sommets auront un poids de 1 par rapport à j_1 et un poids de 0 par rapport à j_2.

Pour mettre en place cette application, on définit au sein de notre fonction de création de cylindre une structure *vertex_weight_parameter* permettant de stocker pour chaque sommet une dépendance par rapport aux différents os. On définit les poids des joints comme expliqué juste au-dessus.

3.3.2 Déformation par skinning

Pour déformer la surface par skinning sur les sommets du maillage, nous devons déterminer la position des points après déformation par skinning à l'instant t. Nous pouvons déterminer la nouvelle position du sommet par la formule suivante :
$p(t) = \Sigma w_j S_j(t) p_0$ avec p_0 la position d'un sommet de coordonnée initiale

Nous avons la matrice de transformation $S_j(t)$ telle que $S_j(t) = T_j(t) B_j^{-1}$ où $T_j(t)$ correspond aux joints du squelette animé dans le repère global et $Bj-1$ correspond à l'inverse des repères de la bind pose.

Pour calculer la matrice de transformation $S_j(t)$, nous devons implémenter deux fonctions :

- *inversed()* qui permet de calculer un squelette dont les joints contiennent les déformations inverses des repères du squelette passé en paramètre. Nous avons donc :

$$\begin{pmatrix} q & t \\ 0 & 1 \end{pmatrix} * \begin{pmatrix} q^{-1} & t^{-1} \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} qq^{-1} & qt^{-1} + t \\ 0 & 1 \end{pmatrix} = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

On peut en déduire que : q = q^{-1} et t = -q^{-1}t

- *multiply()* qui permet de calculer le produit des déformations de chaque joint des squelettes passés en paramètres. On retrouve la formule suivante vu précédemment:

$$q_j = q_{p(j)} * q$$
$$t_j = q_{p(j)} * t + t_{p(j)}$$

Une fois la matrice de déformation obtenue, on crée une fonction *apply_skinning* qui réalise cette déformation sur les sommets du maillage. Nous appliquons donc ces fonctions à nos points pour effectuer la transformation. Nous implémentons la somme des poids de skinning, la matrice de transformation et les positions initiales des sommets pour déterminer les nouvelles positions des sommets.

On observe une déformation du cylindre qui suit la géométrie du squelette mais une dépendance rigide entre chaque sommet et un unique os donnant une déformation importante à l'endroit de la pliure. Cela est dû au fait que nous avons donné des valeurs fixes aux poids de skinning. Nous allons donc modifier les poids de skinning pour obtenir une déformation plus lisse.

3.3.3 Poids d'interpolations

Pour obtenir une déformation plus réaliste du cylindre entre chaque sommet et un unique os, nous allons modifier les poids de skinning afin d'obtenir une déformation lisse, c'est-à-dire que nous allons interpoler ces poids. L'influence du joint 0 décroit linéairement (entre 1 et 0) et l'influence du joint 1 croît linéairement (entre 0 et 1) en fonction de la distance le long de l'axe du cylindre.
Nous obtenons le résultat suivant:

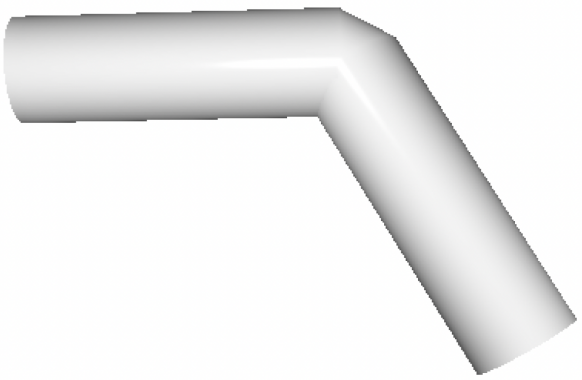

Figure 4: Déformation de la surface du cylindre par skinning avec modification des poids de skinning

3.4 Interpolation de repères

Nous avons animé notre cylindre en fonction de 4 poses clés qui effectuaient un saut de 30 degrés à chaque changement, voir figure .. . Cette animation est en réalité grossière et donnerait un mauvais rendu. Pour pouvoir avoir une meilleure animation de notre cylindre, nous allons interpoler la position des repères entre deux instants t_i et t_{i+1}. Soit le paramètre $\alpha = \frac{(t-t_i)}{(t_{i+1}-t_i)}$ (compris entre 0 et 1) qui permet cette interpolation de la position et des orientations des joints entre deux frames.
L'expression de l'interpolation d'une position p variant entre une position p_1 et une position p_2 en fonction de alpha est : $p = (1-\alpha)p_1 + \alpha p_2$
De même, l'expression de l'interpolation des orientations suivants les quaternions q entre q_1 et q_2 est exprimée cette fois-ci avec la fonction slerp dans *quarternion.hpp* correspondant à une méthode d'interpolation linéaire sphérique.

Nous obtenons le résultat suivant:

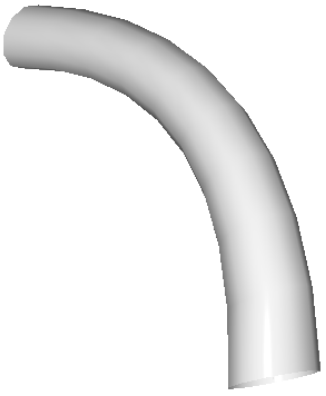

Figure 5: Déformation de la surface avec interpolation de la position des repères entre deux instants t_i et t_{i+1}

On remarque une amélioration de la qualité de l'animation au niveau de la continuité du mouvement.

V. Skinning d'un monstre

Après avoir réalisé l'animation d'un cylindre composé de son squelette, nous allons appliquer la méthode de skinning sur un personnage. Nous allons utiliser une démarche similaire à celle que nous venons d'implémenter sur le cylindre pour

pouvoir visualiser l'animation du personnage. Nous remarquons tout de suite que le skinning du monstre est plus complexe que le skinning du cylindre. C'est pourquoi nous allons récupérer les informations nécessaires au fonctionnement du monstre qui sont déjà établis dans le répertoire *data/*. Ces informations sont le mesh du monstre, le bind pose skeleton, le vecteur parents id, la texture du monstre et son animation. Nous allons tout d'abord afficher le mesh et le squelette associé au monstre puis l'animation du squelette afin de vérifier la cohérence des résultats. L'animation du squelette fonctionnant bien, nous pouvons appliquer la méthode skinning au monstre. Nous obtenons les résultats suivant:

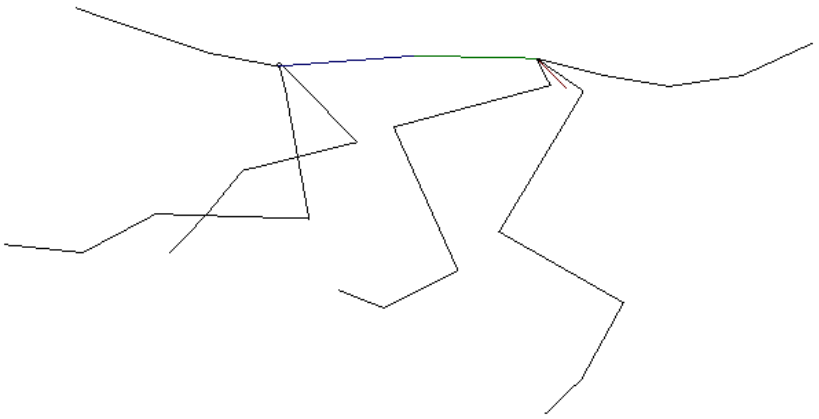

Figure 6: Squelette du monstre

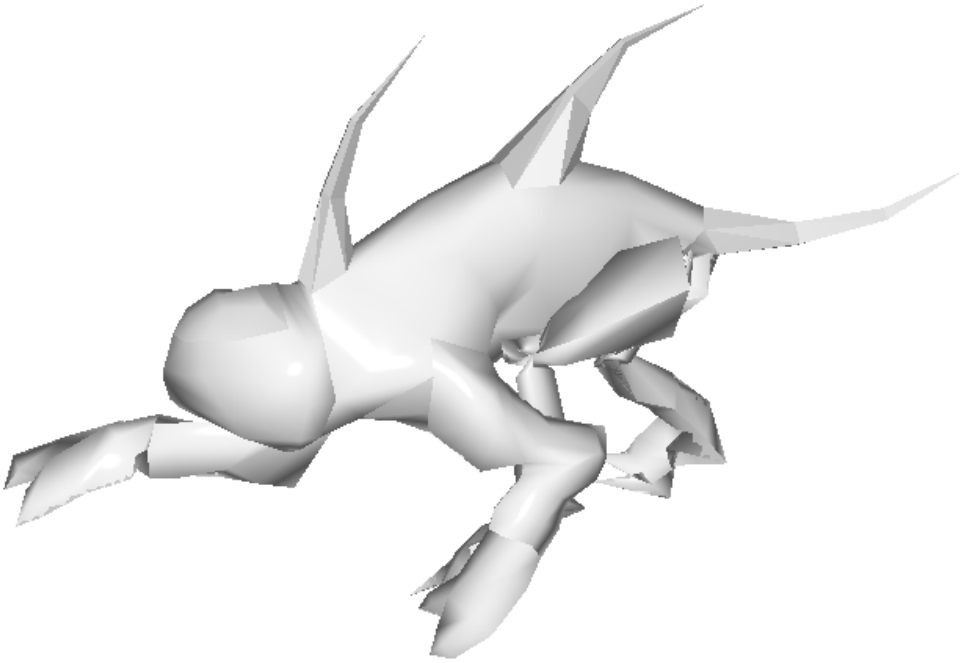

Figure 7: Skinning du monstre

VI. Conclusion

Nous avons implémenté pendant ce TP, une méthode d'animation par skinning pour modéliser la déformation d'un cylindre et appliquer la déformation sur un personnage complet chargé à partir d'un fichier. La méthode d'animation par skinning consiste alors à associer un squelette formé de plusieurs os reliés par des joints s'articulant entre eux, au maillage de la surface à animer. Les positions respectives de chaque joint sont définies dans le repère global et en fonction des joints parents, il faut donc définir l'orientation et la position des joints à chaque instant considéré. Il est nécessaire de respecter une certaine hiérarchie de ces joints. Pour animer le squelette, nous avons défini la position et l'orientation des joints à chaque instant considéré. Nous avons créé 4 poses pour animer le squelette, bouclant après un certain nombre de frames. Pour déformer la surface de manière à ce qu'elle suive le mouvement du squelette, nous avons utilisé les poids de skinning associés à chaque sommet du maillage de la surface pour interpoler l'attachement des sommets aux os alentours. Pour rendre l'animation plus réaliste, nous avons interpolé la position des repères entre deux instant pour donner une impression de continuité. Finalement, nous avons pu utiliser cette méthode de skinning sur un personnage dont les données étaient déjà connues. Nous avons implémenté une méthode de skinning fonctionnel et efficace.