Skinning
CPE

5ETI IMI
TP original de Damien Rohmer

Figure 1: Exemple d'un monstre dont l'animation est géré par la méthode de skinning.

1 But

L'objectif de ce TP est de coder une déformation d'objet 3D par la méthode dite du skinning consistant à déformer une surface en suivant un squelette articulé guidant différentes parties de l'objet.

- Dans un premier temps, nous générerons la déformation d'un objet simple de type cylindre formé de deux os.
- Dans un second temps, nous appliquerons la déformation sur un personnage complet chargé à partir d'un fichier (voir figure 1).

2 Prise en main de l'environnement

2.1 Compilation

Question 1 *Compilez le projet et assurez-vous que celui-ci s'exécute (fenêtre interactive affichant un plan 3D). Assurez-vous que vous puissiez l'éditer depuis l'éditeur de votre choix (QtCreator ou autre). N'oubliez pas que vous pouvez configurer l'emplacement du lancement de votre exécutable dans vos IDE.*

2.2 Les différents répertoires des sources

Vous retrouverez les répertoires ayant déja servis dans d'autres projets tels que `lib/` et `image/`. Dans ce TP, nous ajoutons le répertoire `skinning` qui contient les classes permettant la gestion d'un squelette d'animation, des poids de skinning et d'un maillage déformé par skinning.

2.3 Scène

Question 2 *Observez la classe* `scene` *qui pour l'instant n'affiche qu'un plan. Notez la partie de chargement des données s'exécutant une seule fois en début de programme et la fonction d'affichage s'exécutant régulièrement.*

3 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 au sommets du cylindre puis déformerons cette surface (flexion du cylindre) en fonction de l'animation du squelette.

3.1 Surface géométrique

Nous allons tout d'abord créer la géométrie d'un cylindre simple.

Vous disposez d'une variable de classe `mesh_cylinder` de type `mesh_skinned`. Cette classe est similaire aux classes `mesh` mais contient en plus les informations de skinning qui vont nous servir plus tard.

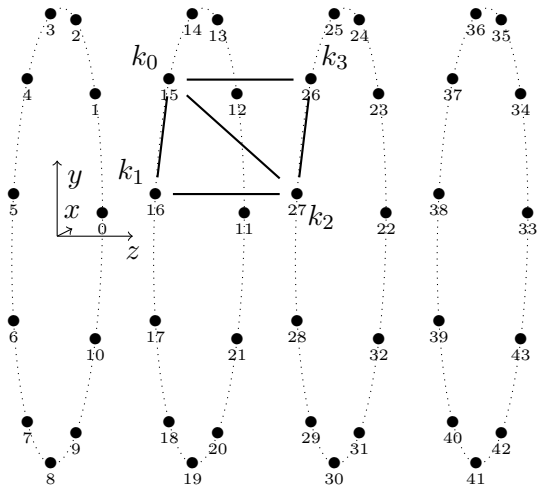

Figure 2: Mesh d'un cylindre orienté sur l'axe z avec $R = 3$, $L = 7.5$, $N_u = 4$, $N_v = 11$, on a $k_0 = (u, v) = u \times N_v + v$.

Question 3 *Créez une fonction dans scène permettant de créer une surface cylindrique orientée suivant l'axe z dont le rayon et la longueur le long de son axe soient passés en paramètre. On passera également le nombre d'échantillons suivant la circonférence et l'axe du cylindre. Le cylindre créé sera de type* `mesh_skinned` *et sera composé de triangles. Notez que vous disposez des méthodes* `add_vertex` *et* `add_triangle_index`*. Notez qu'il est important de prévoir des sommets le long de l'axe du cylindre afin de pouvoir déformer celui-ci par la suite.*

Question 4 *Dans la fonction* `load_scene()`*, appelez votre fonction de création de cylindre afin de compléter la variable* `mesh_cylinder`*. On pourra considérer un cylindre de rayon* 4 *et de longueur* 50*.*

Question 5 *Similairement à la variable* `mesh_ground`*, appelez la méthode* `fill_empty_field_by_default()` *afin de complétez les champs non remplis (textures, couleurs,*

normales). Puis envoyez les données sur le GPU par le biais de la variable `mesh_cylinder_opengl` *et de la méthode* `fill_vbo()`.
Au final, votre code pourra ressembler à cela

```
float const length = 50.0f;
float const radius = 4.0f;
mesh_cylinder = build_cylinder(radius,length,40,40);
mesh_cylinder.fill_empty_field_by_default();
mesh_cylinder_opengl.fill_vbo(mesh_cylinder);
```

Question 6 *Dans la fonction d'affichage, demandez l'affichage de votre cylindre.*

```
mesh_cylinder_opengl.draw();
```

Assurez-vous qu'il apparaisse bien à l'écran.

3.2 Squelette du cylindre

3.2.1 Position de repos

Nous allons désormais 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.

Nous allons considérer que le cylindre est formé d'une hiérarchie simple de deux os mis bout à bout. Nous allons donc définir 3 positions permettant de définir ces deux os, chaque position étant associée à un repère propre. Le squelette souhaité est illustré en figure 3.2.1.

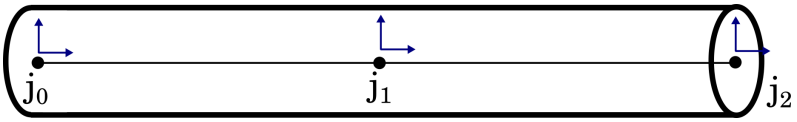

Figure 3: Géométrie du squelette du cylindre.

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

1. Une première structure `skeleton_parent_id` va stocker la connectivité de la hiérarchie. Ici, on pourra considérer que: parent(0)=-1, parent(1)=0, et parent(2)=1.

2. Une seconde structure géométrique qui va stocker chaque repère de joint à l'indice correspondant. Classiquement, 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.

Question 7 *Quelle est l'expression (en tant que translation et rotation) du repère du joint j_1 par rapport au repère du joint j_0. Idem pour j_2 par rapport à j_0. On supposera pour simplifier que les repères sont alignés et ne subissent aucune rotation d'un joint par rapport au suivant.*

Question 8 *Implémentez une fonction permettant d'initialiser ces deux squelettes que l'on nommera respectivement* `sk_cylinder_parent_id`, *et* `sk_cylinder_bind_pose`. *Rem. Un quaternion exprimant une rotation d'angle 0 degré peut être initialisé par les valeurs suivantes* `quaternion(0,0,0,1)`.

Comme la position et l'orientation des joints sont exprimées dans un repère local, il n'est pas possible de visualiser le squelette immédiatement. Il faut au préalable convertir ces coordonnées locales en coordonnées globales.

Notons G_j la matrice correspondante aux coordonnées du repère j exprimé suivant des coordonnées globales. Notons L_j la matrice correspondante à ce même repère exprimé suivant les coordonnées locales par rapport au joint parent. Notons $p(j)$ l'indice du parent du joint j. On a alors la relation récursive

$$\mathrm{G}_j = \mathrm{G}_{p(j)}\,\mathrm{L}_j\ .$$

On rappel également qu'une matrice M de taille 4×4 exprimant une transformation isométrique peut se décomposer de la manière suivante

$$\mathrm{M} = \left(\begin{array}{c|c} q & t \\ \hline 0 & 1 \end{array}\right)\ ,$$

avec q la rotation (exprimée comme une matrice 3x3) et t la translation (vecteur à 3 dimensions). Dans notre cas, la rotation est exprimée sous la forme d'un quaternion.

Question 9 *Donnez 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.*

Question 10 *Complétez la fonction* `local_to_global()` *du fichier* `skeleton_geometry` *permettant d'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.*
Vérifiez votre résultat sur le squelette du cylindre en vous assurant que vous ayez les valeurs attendues.

Pour afficher le squelette, il est nécessaire d'afficher des segments joignant deux joints consécutifs dans la hiérarchie (ce que l'on appelle un os du squelette). La fonction `extract_bones()` vient générer un vecteur stockant ces os sous la forme de paire de points 3D.

Question 11 *Complétez la fonction* `extract_bones()`.

Question 12 *Appelez la fonction* `draw_skeleton()` *dans l'affichage de votre scène en l'appliquant sur le vecteur contenant la position des os. Assurez vous que vous puissiez visualiser votre squelette. À ce moment, votre code pourra ressembler à cela dans la fonction d'affichage*

```
skeleton_geometry const sk_cylinder_global =
  local_to_global(sk_cylinder_bind_pose,sk_cylinder_parent_id);
std::vector<vec3> const sk_cylinder_bones =
  extract_bones(sk_cylinder_global,sk_cylinder_parent_id);
draw_skeleton(sk_cylinder_bones);
```

Note: L'affichage du squelette doit se faire préférentiellement pendant que le shader spécifique au squelette est activé (c.a.d après la ligne `setup_shader_skeleton(shader_skeleton)`).

3.2.2 Animation du squelette

Nous allons désormais créer la géométrie d'un squelette animée. Cela consiste à définir la position et l'orientation des joints à chaque instant considérés.

On souhaite réaliser l'animation de la figure 3.2.2 avec les angles respectifs de 0, 30, 60 et 90 degrés.

La structure `skeleton_animation` permet de stocker une suite de squelette géométrique. On y exprimera également les coordonnées des repères des joints de manière locale.

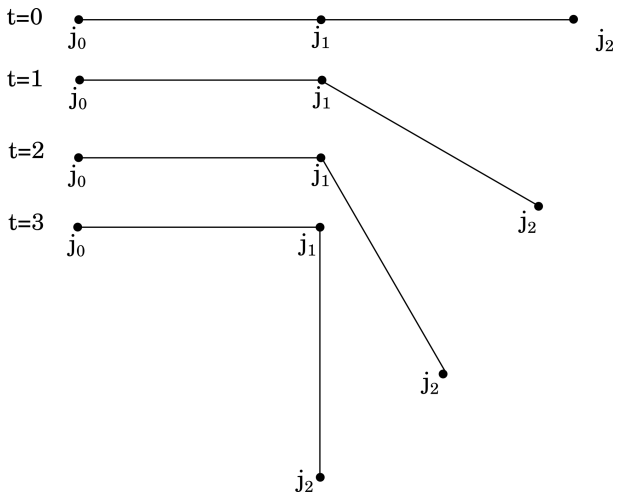

Figure 4: Animation du squelette du cylindre pour 4 poses clés.

Question 13 *Créez une fonction permettant de remplir votre squelette d'animation* `sk_cylinder_animation` *avec l'animation décrite en figure 3.2.2.*

Question 14 *Visualisez les 4 positions de votre animation (en remplaçant la visualisation de la* bind pose *par celle de votre squelette animé).*

Question 15 *Mettez en place une transition automatique permettant de visualiser l'animation de votre squelette en boucle. Pour cela, vous définirez une variable indiquant la pose courante (entre 0 et 3) qui sera affichée. Cette variable s'incrémentera toute les x millisecondes (et reviendra à 0 une fois l'animation terminée). Pour réaliser un compteur, vous disposez par exemple de la classe* `QTime` *de* `Qt` *et de ses méthodes* `elapsed()` *et* `restart()`.

3.3 Déformation de la surface

3.3.1 Attribution des poids de skinning

Nous allons déformer la surface de manière à ce qu'elle suive le mouvement du squelette. La première étape consiste à attribuer les poids de skinning, c'est à dire à définir l'attachement entre chaque sommet de la surface et les repères du squelette.

Dans notre cas, seuls les deux premiers joints vont nous servir (le troisième n'ayant qu'un intérêt visuel lors de l'affichage du squelette).

Considérons que la première moitié du cylindre suive rigidement le premier os, et la seconde moitié le deuxième os. Cela signifie que pour la première moitié du cylindre, les sommets ont un poids de 1 par rapport au joint 0, et un poids de 0 par rapport aux autres joints. Et pour la seconde moitié du cylindre, les sommets ont un poids de 1 par rapport au joint 1, et un poids de 0 par rapport aux autres joints.

Question 16 *Mettez en place l'application de ces poids de skinning lors de la création de votre cylindre. Notez que la structure* `vertex_weight_parameter` *peut stocker pour chaque sommet une dépendance à 6 os différents. Tous les autres étant considéré comme ayant un poids de 0.*

3.3.2 Déformation par skinning

Soit un sommet de coordonnées initiale p_0 et de coordonnées $p(t)$ après déformation par skinning à l'instant t. On rappelle que

$$p(t) = \sum_{\text{joints}j} \omega_j \, \mathrm{S}_j(t) \;\; p_0 \, ,$$

avec S_j la matrice de transformation telle que

$$S_j(t) = \mathrm{T}_j(t) \, \mathrm{B}_j^{-1} \, ,$$

où $\mathrm{T}_j(t)$ est la transformation du joint j à l'instant courant t exprimée dans le repère globale, et B_j^{-1} est la transformation inverse du joint j exprimée dans le repère global.

Notons que $\mathrm{T}_j(t)$ correspond dans notre cas aux joints du squelette animé exprimée dans le repère global, et B_j^{-1} correspond à l'inverse des repères de la *bind pose* exprimée dans le repère global.

Question 17 *Complétez la fonction* `inversed()` *du fichier* `skeleton_geometry` *permettant de calculer un squelette dont les joints contiennent les déformations inverses des repères du squelette passé en paramètre.*

Question 18 *Complétez la fonction* `multiply()` *du fichier* `skeleton_geometry` *permettant de calculer le produit des déformations de chaque joints des squelettes passés en paramètres.*

On suppose finalement que le calcul de la déformation par skinning est réalisé par le code suivant (dans la fonction d'affichage de la scène)

```
skeleton_geometry const sk_cylinder_inverse_bind_pose =
   inversed(sk_cylinder_bind_pose);
skeleton_geometry const sk_cylinder_binded =
   multiply(sk_cylinder_global,sk_cylinder_inverse_bind_pose);
mesh_cylinder.apply_skinning(sk_cylinder_binded);
mesh_cylinder.fill_normal();
mesh_cylinder_opengl.update_vbo_vertex(mesh_cylinder);
mesh_cylinder_opengl.update_vbo_normal(mesh_cylinder);
mesh_cylinder_opengl.draw();
```

Question 19 *Complétez la fonction* `apply_skinning()` *permettant de réaliser la déformation par skinning sur les sommets du maillage. Notez que vous avez toujours accès aux sommets originaux du maillage dans la variable* `vertices_original_data`*.*
Vérifiez que votre cylindre se déforme bien en suivant le mouvement du squelette (voir figure 3.3.2).

3.3.3 Poids d'interpolations

Le skinning actuel donne une dépendance rigide entre chaque sommet et un unique os ce qui donne une déformation importante à l'endroit de la pliure.

Question 20 *Modifiez les poids de skinning de manière à obtenir une déformation lisse. Par exemple, on pourra considérer que l'influence du joint 0 décroit linéairement (entre 1 et 0) et l'influence du joint 1 croit linéairement (entre 0 et 1) en fonction de la distance le long de l'axe du cylindre.*

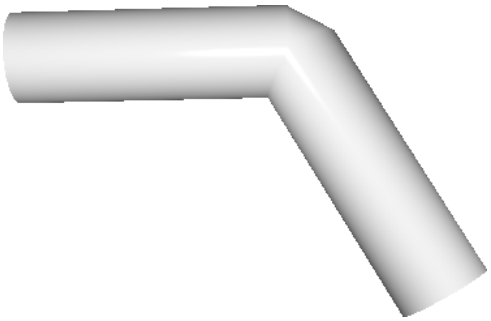

Figure 5: Déformation du cylindre par skinning. Le joint 1 est orienté à 60 degrés par rapport au joint 0.

3.4 Interpolation de repères

L'animation de votre cylindre est très grossière, car celle-ci ne contient que 4 poses clés qui font un saut de 30 degrés à chaque changement. Il est possible de donner une impression de continuité en interpolant la position des repères entre deux instant. Soit l'instant t compris entre les instants discrets t_i, et t_{i+1}, et considérons le paramètre $\alpha = (t - t_i)/(t_{i+1} - t_i)$. Ce paramètre α compris entre 0 et 1 nous donne un paramètre permettant d'interpoler la position et les orientations des joints entre deux frames.

Question 21 *Soit deux positions p_1 et p_2. On souhaite interpoler en fonction du paramètre α une position p variant linéaire entre p_1 et p_2. Donnez l'expression de p en fonction de α, p_1 et p_2.*
De la même manière, on souhaite interpoler les orientations suivant les quaternions q entre q_1 et q_2. Quelle méthode utilises t-on ?

Question 22 *Complétez la méthode* `interpolated()` *de la classe* `skeleton_geometry` *afin de calculer l'interpolation entre deux poses de squelettes fixes en fonction du paramètre α.*

La méthode

```
skeleton_animation::operator()(int const frame,float const alpha)
```

fait appel à cette interpolation, et permet ainsi de calculer directement un squelette correspondant à l'interpolation pour un instant donné entre deux poses clés.

Question 23 *Faite appel à cette méthode dans votre programme pour obtenir une animation continue de votre squelette (voir exemple en figure 3.4).*

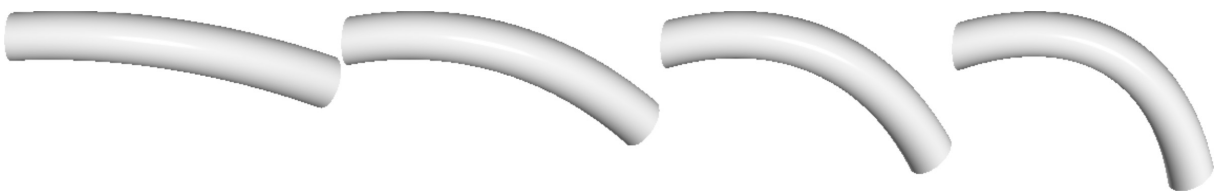

Figure 6: Déformation continue du cylindre obtenue par interpolation du squelette.

4 Skinning d'un monstre

Les données correspondant au skinning d'un monstre vous sont fournies dans le répertoire `data/`. Le fichier `.obj` contient les poids de skinning associé au squelette donné dans les fichier. Une animation vous est également fournie. Ces données ont été générées à l'aide d'un logiciel permettant à un artiste numérique de *peindre* manuellement des poids de skinnings (ex. Blender) puis celles-ci ont été sauvegardées sous les formats de fichiers dont vous disposez.

Question 24 *En utilisant une démarche similaire à celle de l'animation du cylindre, visualisez l'animation du monstre. Procédez de manière méthodique (visualisez la géométrie au repos, le squelette au repos, le squelette animé, et enfin déformez la surface du monstre).*

Question 25 *Visualisez les poids de skinning en couleurs sur la surface du monstre en fonction des os.*

5 Extensions

Question 26 *Ajoutez un déplacement dans la scène.*

Question 27 *Améliorez l'interface de manière à afficher ou non les informations que vous souhaitez (position de repos, squelette, maillages, etc).*

Question 28 *Réfléchissez à une implémentation de la méthode de skinning sur GPU. Que passez-vous en paramètre uniforme, quelles données doivent être envoyées en même temps que les sommets. Tentez d'implémenter une telle déformation sur GPU.*