Développement d'applications interactives

Compte-rendu TP2

Animation

Louis Jean
Master 2 IMAGINE
Université de Montpellier
N° étudiant : 21914083

18 septembre 2024

Table des matières

1 Introduction

Le but de ce TP est de calculer les poids de skinning linéaires à partir d'un squelette et d'une animation donnés, puis d'appliquer ces poids pour animer un maillage en utilisant des transformations squelettiques. Enfin, nous implémentons la cinématique inverse en utilisant la méthode du Cyclic Coordinate Descent (CCD) pour manipuler le squelette de manière interactive.

2 Calcul des poids

Les poids de skinning déterminent l'influence de chaque os sur les sommets du maillage. Pour les calculer, nous avons utilisé une méthode basée sur la distance euclidienne entre un sommet et les os du squelette.

2.1 Méthode

Pour chaque sommet v_i du maillage, nous calculons son poids par rapport à chaque os b_j en utilisant la formule suivante :

$$w_{ij} = \frac{\left(\frac{1}{d_{ij}}\right)^n}{\sum_{k=1}^{B}\left(\frac{1}{d_{ik}}\right)^n}$$

où d_{ij} est la distance minimale entre le sommet v_i et l'os b_j, n est un exposant qui contrôle la répartition des poids et B est le nombre total d'os que comporte le squelette du maillage.

2.2 Implémentation

Voici le code que nous avons implémenté :

```
void Mesh::compute_skinning_weights(Skeleton & skeleton) {
    int nbVertices = V.size();
    int nbBones = skeleton.bones.size();
    double n = 5.0;
    for(int i = 0; i < nbVertices; ++i) {
        MeshVertex &vertex = V[i];
        Vec3 p = vertex.p;
        double sum = 0.0;
        for(int j = 0; j < nbBones; ++j) {
            Bone &bone = skeleton.bones[j];
            Vec3 p0 = skeleton.articulations[bone.joints[0]].p;
            Vec3 p1 = skeleton.articulations[bone.joints[1]].p;

            Vec3 p0p = p - p0;
            Vec3 p0p1 = p1 - p0;
            float t = Vec3::dot(p0p, p0p1) / Vec3::dot(p0p1, p0p1);
            float d;
            if(t < 0) {
                d = p0p.length();
            }
            else if(t > 1) {
                d = (p - p1).length();
            }
            else {
                Vec3 proj = p0 + t * p0p1;
                d = (p - proj).length();
            }
            double weight = pow(1.0 / d, n);
            sum += weight;
```

```
            vertex.w.push_back(weight);
        }
        for(int j = 0; j < nbBones; ++j) {
            vertex.w[j] /= sum;
        }
    }
}
```

Listing 1: Calcul des poids de skinning

2.3 Résultats

Nous avons testé cette méthode avec deux valeurs différentes de n : $n = 2$ et $n = 5$.

Figure 1: Visualisation des poids de skinning pour $n = 2$ (gauche) et $n = 5$ (droite)

- Pour $n = 2$: Les poids sont répartis de manière plus lisse entre les os. Chaque sommet est influencé par plusieurs os, ce qui produit des déformations plus douces.
- Pour $n = 5$: Les poids sont concentrés autour des os les plus proches. Cela réduit l'influence des os lointains, ce qui rend les déformations plus rigides et précises autour des articulations, mais peut introduire des cassures visibles lors de l'animation.

3 Mise à jour des transformations et des normales

Après avoir calculé les poids, nous appliquons les transformations squelettiques aux sommets du maillage en utilisant le skinning linéaire.

3.1 Méthode

Pour chaque sommet v_i, sa position transformée v_i' est calculée en faisant la somme pondérée des transformations appliquées par chaque os :

$$v_i' = \sum_{j=1}^{B} w_{ij} \left(R_j v_i + T_j \right)$$

où : R_j est la rotation du monde pour l'os b_j, T_j est la translation du monde pour l'os b_j, w_{ij} est le poids du sommet v_i pour l'os b_j.
Les normales sont également mises à jour en appliquant les rotations correspondantes.

3.2 Implémentation

Le code suivant réalise cette transformation :

```
void Mesh::drawTransformedMesh(SkeletonTransformation & transfo) const {
    std::vector<Vec3> new_positions(V.size());
    std::vector<Vec3> new_normals(V.size());

    for(unsigned int i = 0; i < V.size(); ++i) {
        Vec3 p = V[i].p;
        Vec3 n = V[i].n;
        int nbBones = V[i].w.size();

        Vec3 sum_position(0.0, 0.0, 0.0);
        Vec3 sum_normal(0.0, 0.0, 0.0);

        for(int j = 0; j < nbBones; ++j) {
            double weight = V[i].w[j];
            BoneTransformation &bt = transfo.bone_transformations[j];

            Vec3 transformed_p = bt.world_space_rotation * p + bt.
   world_space_translation;
            Vec3 transformed_n = bt.world_space_rotation * n;

            sum_position += weight * transformed_p;
            sum_normal += weight * transformed_n;
        }

        new_positions[i] = sum_position;
        new_normals[i] = sum_normal.normalized();
    }

    // Code pour dessiner le maillage avec les nouvelles positions et normales...
}
```

Listing 2: Transformation des sommets et des normales

3.3 Résultats

En appliquant ces transformations, le maillage suit les mouvements du squelette de manière fluide. Les normales mises à jour garantissent que l'éclairage reste cohérent avec les nouvelles positions des sommets, même si c'est compliqué de vraiment voir une différence en raison de la nature du maillage en lui-même.

En augmentant la valeur de n, nous observons que les zones proches des articulations deviennent plus rigides, ce qui est visible lors des mouvements du squelette. Avec des valeurs de n plus faibles, les déformations sont plus lisses.

4 Cinématique inverse

La cinématique inverse permet de contrôler le squelette en fixant la position d'une articulation cible, les autres articulations s'adaptant pour atteindre cette position.

4.1 Méthode du Cyclic Coordinate Descent (CCD)

Le CCD est une méthode itérative qui ajuste les rotations des articulations en partant de l'extrémité du membre et en remontant jusqu'à la racine, en orientant chaque articulation pour rapprocher l'extrémité du membre de la position cible.

4.2 Implémentation

Voici la fonction que nous avons implémentée :

```
void updateIKChain(SkeletonTransformation & transfoIK, unsigned int
    targetArticulation, Vec3 targetPosition, unsigned int maxIterNumber = 20,
    double epsilonPrecision = 1e-6) {
     if (targetArticulation == -1) {
         return;
     }

     for (unsigned int iter = 0; iter < maxIterNumber; iter++) {
         unsigned int currentArticulation = targetArticulation;

         while (articulations[currentArticulation].fatherBone != -1) {
             unsigned int fatherBone = articulations[currentArticulation].
    fatherBone;
             unsigned int fatherArticulation = bones[fatherBone].joints[0];

             Vec3 E_position = transfoIK.articulations_transformed_position[
    currentArticulation];
             Vec3 R_position = transfoIK.articulations_transformed_position[
    fatherArticulation];

             Vec3 RE = E_position - R_position;
             Vec3 RD = targetPosition - R_position;

             Mat3 rotation = Mat3::getRotationMatrixAligning(RE, RD);
             BoneTransformation &bt = transfoIK.bone_transformations[fatherBone];

             bt.localRotation = rotation * bt.localRotation;

             computeGlobalTransformationParameters(transfoIK);

             currentArticulation = fatherArticulation;
         }

         double distance = (transfoIK.articulations_transformed_position[
    targetArticulation] - targetPosition).length();
         if (distance < epsilonPrecision) {
             break;
         }
     }
}
```

Listing 3: Implémentation du CCD pour la cinématique inverse

4.3 Résultats

En utilisant cette implémentation, nous pouvons déplacer l'articulation cible en utilisant les touches du clavier, et le squelette s'adapte en temps réel pour atteindre la position souhaitée. Le maillage suit le mouvement grâce au skinning linéaire précédemment implémenté.

5 Conclusion

Ce TP nous a permis de comprendre et d'implémenter les concepts fondamentaux de l'animation de personnages, notamment le skinning linéaire et la cinématique inverse. Nous avons observé l'impact des paramètres sur le comportement du maillage animé et mis en œuvre une méthode efficace pour contrôler le squelette de manière interactive.