Modélisation et géométrie discrète

Compte-rendu TP10

Animation de squelette

Louis Jean
Master 1 IMAGINE
Université de Montpellier

5 décembre 2023

Table des matières

1 Introduction

Ce compte-rendu porte sur le travail réalisé dans le cadre du TP sur la création d'animations par interaction avec un modèle articulé, en se focalisant particulièrement sur le cycle de la marche d'un personnage.
Le TP se divise en deux grandes parties : la première consiste à implémenter divers types de déplacements pour un modèle articulé – déplacement simple, snake-like, rigide, et rotation – en tenant compte de paramètres tels que la force et le sens du mouvement. La seconde partie requiert une réflexion approfondie sur la génération d'images intermédiaires pour créer un cycle de marche fluide et réaliste, en partant d'une série d'images clés.
La base de code fournie est en JavaScript. Elle implémente un squelette composé de noeuds, sur lesquels on peut agir avec la souris, en choisissant au préalable dans le code le type de mouvement souhaité.

2 Interaction avec un modèle articulé

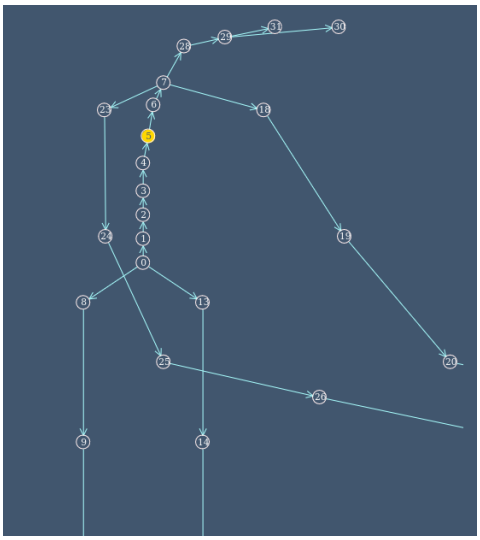
Dans cette section, nous explorons plusieurs méthodes de déplacement pour composer des animations.

2.1 Mouvement snake-like

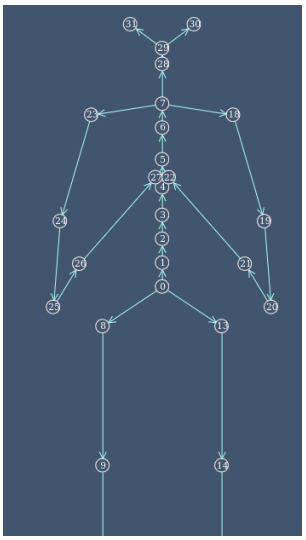
Le mouvement de type 'serpent' (snake-like) autorise le déplacement coordonné de plusieurs sommets connectés, initié par l'interaction avec un seul sommet. Ce type de déplacement préserve la longueur des liens entre les sommets, ainsi que la direction et l'intensité du mouvement, tout en induisant une modification des angles formés par ces liens.
Malheureusement, je n'ai pas réussi à bien implémenter cette fonctionnalité, et j'ai obtenu un résultat assez particulier.

```
1  class SnakeMove extends CMove {
2      initMove(p_e, p_previousLocation, p_node, d) {
3          this.displayer = d;
4          let delta = new Coord2D(p_e.offsetX -
               ↪ p_previousLocation.x, p_e.offsetY -
               ↪ p_previousLocation.y);
5          p_node.moveOnCanvas(delta, d);
6          let edges = p_node.getEdges(this.sens);
7          for(let edge of edges) {
8              let connectedNode = this.sens == Sens.forward
                   ↪  ? edge.to : edge.from;
9              let connectedDelta = new Coord2D(delta.x,
                   ↪ delta.y);
10             for(let i = 0; i < this.force; i++) {
11                 this.initMove(p_e,p_previousLocation,
                       ↪ connectedNode,d);
12             }
13         }
14     }
15 }
```

Figure 1: Tentative d'implémentation du mouvement snake-like



(a) Rendu après avoir agi sur le noeud 5



(b) Rendu après avoir agi sur les noeuds 18 et 23

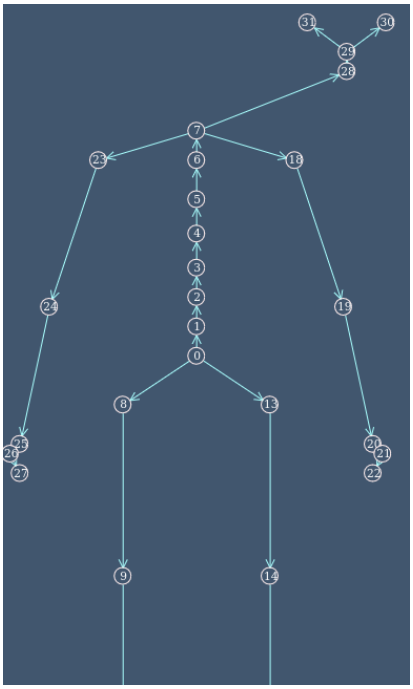
Figure 2: Déplacements induits par le code ci-dessus

2.2 Mouvement rigide

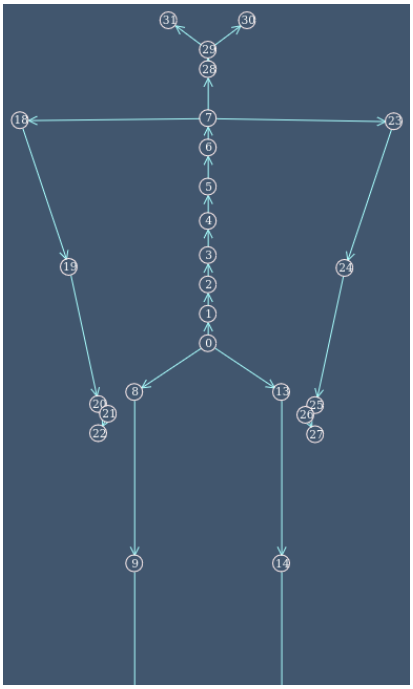
Le mouvement 'rigide' facilite le déplacement simultané de plusieurs sommets connectés, déclenché par l'interaction avec un sommet unique. Ce type de mouvement maintient non seulement la longueur et l'orientation des liens ainsi que l'intensité du mouvement, mais préserve également les angles initiaux entre les liens, garantissant ainsi la rigidité structurelle du modèle.

```
1  class BlocMove extends CMove {
2      initMove(p_e, p_previousLocation, p_node, d) {
3          this.displayer = d;
4          let delta = new Coord2D(p_e.offsetX -
               ↪ p_previousLocation.x, p_e.offsetY -
               ↪ p_previousLocation.y);
5          p_node.moveOnCanvas(delta, d);
6          let edges = p_node.getEdges(this.sens);
7          for(let edge of edges) {
8              let connectedNode = this.sens == Sens.forward
                   ↪  ? edge.to : edge.from;
9              let connectedDelta = new Coord2D(delta.x,
                   ↪ delta.y);
10             this.initMove(p_e,p_previousLocation,
                   ↪ connectedNode,d);
11         }
12     }
13 }
```

Figure 3: Implémentation du mouvement rigide



(a) Rendu après avoir agi sur le noeud 28



(b) Rendu après avoir agi sur les noeuds 18 et 23

Figure 4: Déplacements induits par un mouvement rigide

2.3 Rotation

La rotation consiste à déplacer un groupe de sommets interconnectés en pivotant autour d'un sommet choisi, en modifiant l'angle d'un lien associé à ce sommet. Ce type de mouvement préserve les longueurs et les angles relatifs des éléments adjacents, tout en tenant compte de la force appliquée et de la direction du mouvement.
J'ai réussi à bien calculer les angles de rotation, mais je ne sais pas pourquoi, aucun mouvement ne se produit lorsque je souhaite déplacer des points. Vous pouvez voir dans la console que lorsque l'on se déplace en suivant un cercle autour d'un noeud, on est bien entre 0 et 2π.

```
class AngleMove extends CMove {
    initMove(p_e, p_previousLocation, p_node, d) {
        this.displayer = d;
        let centerNodePos = p_node.getCoordShared(0);
        let mousePos = new Coord2D(p_e.offsetX, p_e.
            ↪ offsetY);
        let angleChange = this.calculateAngleChange(
            ↪ mousePos, centerNodePos);
        let edges = p_node.getEdges(this.sens);
        for (let edge of edges) {
            let connectedNode = this.sens == Sens.forward
                ↪  ? edge.to : edge.from;
            this.rotateNodes(connectedNode, angleChange);
        }

    }
    calculateAngleChange(mousePos, centerNodePos) {
        let directionVector = Coord2D.vecteur(
            ↪ centerNodePos, mousePos);
        let angleChange = directionVector.alpha();
        console.log("Angle : ",angleChange);
        return angleChange;
    }
    rotateNodes(centerNode, angleChange, d) {
        let edges = centerNode.getEdges();
        for (let edge of edges) {
            let connectedNode = this.sens == Sens.forward
                ↪  ? edge.to : edge.from;
            let currentPos = connectedNode.getCoordShared
                ↪ (0);
            let rotatedPos = currentPos.rotateZ(
                ↪ angleChange, centerNode.getCoordShared
                ↪ (0));
            console.log(rotatedPos);
            let delta = Coord2D.soustraction(rotatedPos,
                ↪ currentPos);
            connectedNode.moveOnCanvas(delta, d);
        }
    }
}
```

Figure 5: Tentative d'implémentation de la rotation

3 Processus de création d'une animation

3.1 Principaux besoins d'un animateur pour la marche articulée

Pour réaliser le cycle de la marche d'un personnage articulé, un animateur a besoin de :

- Identification des images clés : des positions clés dans le cycle de la marche.
- Manipulation des articulations : ajuster les angles des articulations pour refléter les mouvements naturels.
- Contrôle de vitesse et de timing : gérer la vitesse de mouvement et assurer un timing précis.
- Visualisation en temps réel : avoir la possibilité de visualiser et ajuster l'animation en cours.
- Symétrie et répétitivité : utiliser des outils pour répliquer ou refléter des mouvements.
- Interpolation : générer des images intermédiaires pour une animation fluide.

3.2 Principes d'animation pertinents pour la marche articulée

Les principes d'animation essentiels pour un cycle de marche réaliste et expressif incluent :

- Anticipation : créer une anticipation avant un mouvement majeur.
- Squash and stretch : appliquer ces effets pour ajouter du dynamisme.
- Timing et espacement : assurer un espacement correct des images pour la fluidité.
- Arcs de mouvement : suivre des trajectoires courbes pour plus de naturel.
- Action secondaire : ajouter des mouvements complémentaires.
- Exagération : accentuer certains aspects selon le style d'animation.
- Pose to pose et straight ahead : planifier les poses clés et compléter avec des animations fluides.
- Solidité : maintenir la cohérence de volume et de structure.

3.3 Structure de données et algorithme pour les images intermédiaires

La création d'images intermédiaires implique :

3.3.1 Structure de données

- Stockage des images clés avec positions et orientations des articulations.
- Enregistrement des moments associés à chaque image clé.
- Données comme la vitesse de marche et la longueur du pas.

3.3.2 Algorithme

- Interpolation linéaire : calculer les positions intermédiaires entre les images clés.
- Pourcentage de progression : déterminer la progression dans le temps.
- Application par articulation : interpoler pour chaque articulation.
- Ajustements spéciaux pour la marche : adapter l'interpolation pour des mouvements naturels.
- Rendu : générer les images intermédiaires après calcul des interpolations.

4 Conclusion

Ce travail pratique sur l'animation de modèles articulés, centré sur le cycle de marche, a été une expérience enrichissante et formatrice. Malgré les défis techniques rencontrés, en particulier avec le mouvement snake-like et la rotation, il a permis une bonne introduction aux techniques d'animation, posant une base solide pour l'exploration future des techniques d'animation avancées.

Merci pour le temps et l'attention que vous avez consacrés à la lecture de ce compte-rendu.