

Computer Animation and Special Effects

Hw2 Forward Kinematics

310551067 吳岱容

- **Introduction**

本次作業主要是實作讀入ASF/AMC格式的檔案，進行輸入骨架動畫的程式編寫，實作內容大約可以分成兩部分：

- Forward Kinematic(在這裡我們讀入建立每個骨頭與關節的資料來編寫動作)
- Motion Warping(利用關鍵影格與插值來呈現動作)

- **Fundamental**

- Global and local coordinate systems

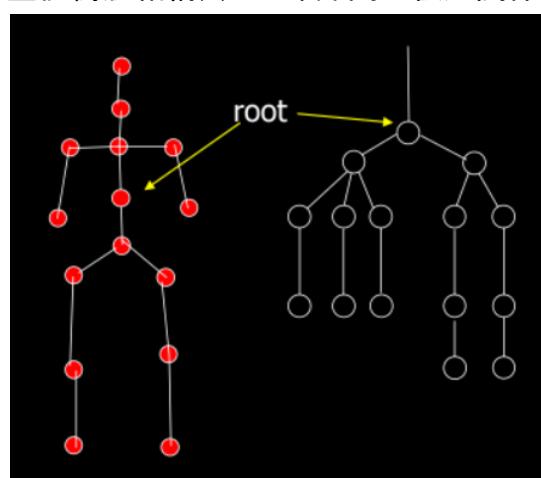
Global coordinate system通常指的是整個系統或架構的座標，相對local coordinate system則是子物件自己擁有的座標系統，每個子物件間的座標軸不一定朝向相同，通常我們會需要一個矩陣來使local system轉換到global system。

- Forward kinematics

Forward kinematics的原理是我們會擁有每個關節相對的位移以及旋轉，然後便利骨架系統從上層(parent)至下層(child)一一計算其旋轉與位移，將其對應到我們的世界座標系統。

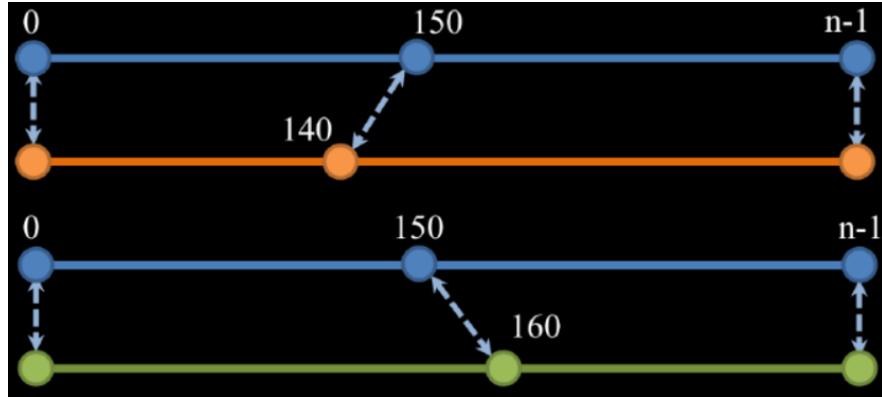
$$\begin{aligned} {}_{i+1}T &= {}_0^1R \left\{ {}_1^2R \cdots {}_{i-2}^{i-1}R \left({}_{i-1}{}^iRV_i + V_{i-1} \right) + V_{i-2} \right] \cdots + V_1 \} \\ {}_{i+1}T &= {}_0^iRV_i + {}_iT \end{aligned}$$

骨架系統是一個類樹狀結構，因此實作Forward Kinematics我們必須遍歷整個樹狀結構，建立節點間的彼此關係。



- Motion Warping

在使用Motion Warping(時間扭曲法), 我們可以藉由建立關鍵影格來調整動畫, 而其他的影格調整可以透過插值法針對位移以及旋轉來做調整。



在部分遊戲引擎內也可以看到類似功能, 能夠透過單一動作資料來產生更多的微調動作。

- Implementation

- Forward kinematics

```
bone->startPosition = Eigen::Vector3f::Zero();
bone->endPosition = Eigen::Vector3f::Zero();
bone->rotation = Eigen::Quaternionf::Identity();

auto R = posture.rotations;
auto T = posture.translations;

bone->startPosition += T[bone->idx];
bone->endPosition = bone->startPosition + bone->direction * bone->length;
bone->rotation = R[bone->idx];
```

針對一開始的root做初始化, 每根骨頭都有兩個點(startPosition與endPosition), 完成長度、朝向、旋轉的設定。

```
if (bone->child != NULL) {
    q.push(bone->child);
    bone = bone->child;
    while (bone->sibling != NULL) {
        q.push(bone->sibling);
        bone = bone -> sibling;
    }
}
```

```

while (!q.empty()) {
    Bone* curr = q.front();

    curr->startPosition = Eigen::Vector3f::Zero();
    curr->endPosition = Eigen::Vector3f::Zero();
    curr->rotation = Eigen::Quaternionf::Identity();

    Eigen::Quaternionf Rm = Eigen::Quaternionf::Identity();

    Rm = (curr->rotationParentCurrent) * R[curr->idx];
    curr->rotation = curr->parent->rotation * Rm;
    curr->startPosition = curr->parent->endPosition;
    curr->endPosition = curr->startPosition + curr->rotation * (curr->direction * curr->length);

    if (curr->child != NULL) {
        q.push(curr->child);
        curr = curr->child;
        while (curr->sibling != NULL) {
            q.push(curr->sibling);
            curr = curr->sibling;
        }
    }
    q.pop();
}

```

遍歷整個骨架系統，針對每一個節點計算朝向、轉向、位移量。

- Motion Warping

```

newMotion.posture(i).translations[j] = Eigen::Vector3f::Zero();
newMotion.posture(i).rotations[j] = Eigen::Quaternionf::Identity();

```

先對位移以及旋轉進行初始化。

```

double key, key1, key2;
if (i <= newKeyframe)
    key = ((double)oldKeyframe) * i / (double)newKeyframe;
else
    key = ((double)totalFrames - 1 - (double)oldKeyframe) * (i - newKeyframe) / (totalFrames - 1 - newKeyframe) +
          oldKeyframe;

```

計算新的Key frame。

```

if (key == (int)key) {
    newMotion.posture(i).translations[j] = motion.posture(key).translations[j];
    newMotion.posture(i).rotations[j] = motion.posture(key).rotations[j];
    continue;
}

```

若key值恰好整數則直接取用對應frame。

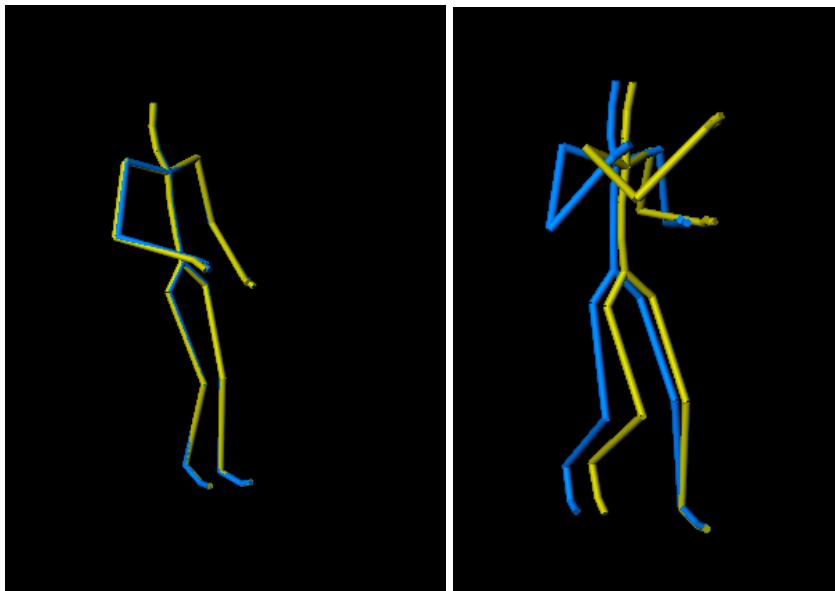
```

//interpolate
key1 = (int)key;
key2 = key1 + 1;
if (key2 >= totalFrames) key2 = key1;
//translation
double a = key - key1;
double b = key2 - key;
newMotion.posture(i).translations[j] =
    (b * motion.posture(key1).translations[j] + a * motion.posture(key2).translations[j]) / (a + b);
//rotation
Eigen::Quaternionf newQ = motion.posture(key1).rotations[j].slerp(a / (a + b), motion.posture(key2).rotations[j]);
newMotion.posture(i).rotations[j] = newQ;

```

若不是則要另外進行插值，在translation我們進行線性插值，而rotation則是spherical插值。

- **Result and Discussion**



藍色部分為原Forward Kinematics的結果，黃色則為Motion Warping之後調整的結果。

一開始兩個模型是重疊的，但是藉由Motion Warping調整的部分漸漸合原動作分開，雖然一樣是揮拳動作，不過時間點跟位置都不太一樣。

透過Motion Warping，我們可以對單一的motion檔案進行編輯微調，從而生出新的動畫。

- **Conclusion**

這次的作業基本上是讓我們讀取ASF/AMC格式的檔案，然後進行動畫的基礎實作，透過這次的作業也確實對於這一部份有更進一步的了解，像是骨架系統在程式方面實際上是如何建立的，還有motion warping在關鍵影格上的應用。