Introduction to Computer Animation HW2 Report

資工系 0416303 楊博凱

- ## Introduction/Motivation

這次作業我們要實作的是透過模擬角色運動在跑步以及接球運動展現 Forward Kinematics 以及 Time Warping。作業中我們會透過已經有的 Function 以及 Eigen 所提供的各種計算達成 rotation matrix, Quaternion, Euler Angles 的轉換與運算。我們也會從 skeleton 的 root 開始計算出各個骨頭間的 start position, end position 以及 rotation 並模擬出整個人型。

- ## Fundamentals
  - Local Coordinate:

    針對每一個骨頭的都存在一個 coordinate，我們設該骨頭的 start position 為原點，並透過 parent 的 rotation 在乘上該骨頭的 Ramc 以及 Rasf 得到在該 local coordinate 的 rotation。運用此 rotation 我們乘上 local direction 以及 normalized 後的 bone length 並加上 start position 就可以得到 end position。以此類推，從 root 開始我們可以 traverse 整個 skeleton。

  - Global Coordinate:

    世界坐標系，以(0, 0, 0)為原點的坐標系，透過此坐標系，我們可以得到各個點、線、面在空間中的位置。然而在模擬作業中的 skeleton 時，若用 global coordinate 計算則會需要諸多轉換，較不方便，因此我們才採用 local coordinate。透過每個 bone 的 local coordinate，我們可以由 start position 以及 rotation 算出 end position。

  - Time Warping:

    Time Warping 是一種對於的時間做縮放的行為，允許我們對於動畫中不同 frame 做時間的改變，例如時間延長或收縮的形式。在作業的第二部分中，由於每個 frame 所花費的時間相同，因此動作沒有快慢之分，若我們想接到球，則需要對於該 skeleton 做 Time Warping 以達到時間縮放的效果，使 skeleton 成功接到球。

  - Forward Kinematics:

    Forward Kinematics 是指在 skeleton 中某 bone 中的關節在指定的時間產生 transition 以及 rotation，而使其之後延伸出去的 bone 受其旋轉的影響而影響。也就是說，當父節點移動或旋轉時，子節點如何受其影響。我們可以透過 AMC File 以及 ASF File 從 Root 開始求出每個時間點的 rotation，並延伸到以其 end position 為 start position 的 local coordinate，求出 skeleton 各個 bone 的位置。

- **Implementation**
  - Forward Kinematics:

這次作業，我一開始先從 joint_spatial_pos[0] vector 讀取進來 Root 的 position 以及 rotation，乘上由 rot_parent_current 儲存的 Rasf 得到最一開始 root 的 rotation 以及 end position。之後透過不同方向運用 for loop 分別求出由 root 延伸出去的四肢以及身體的 skeleton position 與 rotation。

$$ {}^{i+1}_{i}R = {}_{i}R_{asf} \cdot {}_{i}R_{amc} $$

```
mat0(0, 0) = this->skeleton().get()->root_bone()->rot_parent_current[0][0];
mat0(1, 0) = this->skeleton().get()->root_bone()->rot_parent_current[1][0];
mat0(2, 0) = this->skeleton().get()->root_bone()->rot_parent_current[2][0];
mat0(0, 1) = this->skeleton().get()->root_bone()->rot_parent_current[0][1];
mat0(1, 1) = this->skeleton().get()->root_bone()->rot_parent_current[1][1];
mat0(2, 1) = this->skeleton().get()->root_bone()->rot_parent_current[2][1];
mat0(0, 2) = this->skeleton().get()->root_bone()->rot_parent_current[0][2];
mat0(1, 2) = this->skeleton().get()->root_bone()->rot_parent_current[1][2];
mat0(2, 2) = this->skeleton().get()->root_bone()->rot_parent_current[2][2];
double x = math::ToRadian(joint_spatial_pos[0][0]), y = math::ToRadian(joint_spatial_pos[0][1]), z = math::ToRadian(joint_spatial_pos[0][2]);
math::RotMat3d_t mat2;
mat2 = math::ComputeRotMatXyz(x, y, z);
mat = mat0 * mat2;
```

從 rot_parent_current 讀取出 ASF File 裡面的 rotation matrix，乘上 Input Vector joint_spatial_pos 中所存的前三個數字 rx, ry, rz 得到該骨頭的 local rotation。

$$ {}^{i}_{0}R = {}^{1}_{0}R\,{}^{2}_{1}R \cdots {}^{i}_{i-1}R $$

從上方 Root 求出的 local rotation 開始 traverse 整個 skeleton 我們可以得到每個 bone 的 global rotation。(然而要特別注意節點的 rotation)

$$ V_i = \hat{V}_i \cdot l_i $$

```
this->skeleton().get()->bone_local_dir(0) * this->skeleton().get()->bone_length(0)));
```

將已經從 global direction 轉換為 local direction 並存在 bone_local_dir ( )中的值乘上已經經過 normalized 的 bone length 我們可以得到 Vi。

$$ {}^{\square}_{i}T = {}^{i-1}_{0}RV_{i-1} + {}_{i-1}{}^{\square}T $$

```
newPose.set_end_pos(newPose.start_pos() + math::Vector3d_t(newPose.rotation() * this->skeleton().get()->bone_local_dir(0) * this->skeleton().get()->bone_length(0)));
```

由每個 bone 的 start position 加上 rotation 乘上 Vi 我們可以得到該 bone 的 end position。並繼續 traverse 下去，計算以此 end position 為 start position 的 bone 的資料。

```
math::RotMat3d_t mat2;
mat2 = math::ComputeRotMatXyz(x, y, z);
Quaterniond q0(mat0);
Quaterniond q2(mat2);
Quaterniond q = q0 * q2;
Quaterniond p;
p.w() = 0;
p.vec() = this->skeleton().get()->bone_local_dir(0) * this->skeleton().get()->bone_length(0);
Quaterniond pRotate = q * p * q.inverse();
mat = q.toRotationMatrix();

newPose.set_rotation(mat);
```

將 Rotation Matrix 轉換成 Quaternion 做運算，而最後存到 bone rotation 中的值我還是再轉回 rotation matrix。Quaternion 的乘法比較不一樣，左邊是乘右邊是除，所以右邊要乘上 inverse。

- Time Warping:

  由於作業中，skeleton 大概是在 frame 為 160 時接到球，但是一開始他的手動得太快了，因此我們要藉由 Time Warping 將前半部時間延長，後半部時間縮短以達到成功接球的效果。因此我選擇將 original frame 的 160 對應到 new frame 的 170，分別對前半段與後半段做 slerp, Linear Interpolation，得到最後可以接到球的 Time Warped Skeleton。

```
int NUM = 170;
math::SpatialTemporalVector6d_t new_motion_seq;
new_motion_seq.Resize(original_motion_sequence_.get()->spatial_size(), original_motion_sequence_.get()->temporal_size());
for (int i = 0; i < original_motion_sequence_.get()->temporal_size(); i++){
    if (i < NUM){
        for (int j = 0; j < original_motion_sequence_.get()->spatial_size(); j++){
            double low = i * 160 / NUM;
            double high = low + 1;
            //std::cout << "i: " << i << ",low: " << low << ",high: " << high << std::endl;
            //std::cout << original_motion_sequence_.get()->element(j, low) << "\n//////////\n";
            //std::cout << original_motion_sequence_.get()->element(j, high) << "\n//////////\n";

            double rxLow = math::ToRadian(original_motion_sequence_.get()->element(j, low)[0]);
            double ryLow = math::ToRadian(original_motion_sequence_.get()->element(j, low)[1]);
            double rzLow = math::ToRadian(original_motion_sequence_.get()->element(j, low)[2]);
            double txLow = original_motion_sequence_.get()->element(j, low)[3];
            double tyLow = original_motion_sequence_.get()->element(j, low)[4];
            double tzLow = original_motion_sequence_.get()->element(j, low)[5];
            double rxHigh = math::ToRadian(original_motion_sequence_.get()->element(j, high)[0]);
            double ryHigh = math::ToRadian(original_motion_sequence_.get()->element(j, high)[1]);
            double rzHigh = math::ToRadian(original_motion_sequence_.get()->element(j, high)[2]);
            double txHigh = original_motion_sequence_.get()->element(j, high)[3];
            double tyHigh = original_motion_sequence_.get()->element(j, high)[4];
            double tzHigh = original_motion_sequence_.get()->element(j, high)[5];
            math::RotMat3d_t mat1;
            mat1 = math::ComputeRotMatXyz(rxLow, ryLow, rzLow);
            Eigen::Quaterniond qLow(mat1);
            math::RotMat3d_t mat2;
            mat2 = math::ComputeRotMatXyz(rxHigh, ryHigh, rzHigh);
            Eigen::Quaterniond qHigh(mat2);

            Eigen::Quaterniond middle = qLow.slerp(0, qHigh);
            math::Vector3d_t middleAxisRadian = math::ComputeEulerAngleXyz(middle.toRotationMatrix());
            math::Vector3d_t middleAxis(math::ToDegree(middleAxisRadian[0]), math::ToDegree(middleAxisRadian[1]), math::ToDegree(middleAxisRadian[2]));
            math::Vector3d_t middlePoint((txLow + txHigh) / 2, (tyLow + tyHigh) / 2, (tzLow + tzHigh) / 2);
            math::Vector6d_t newElement(middleAxis, middlePoint);
            new_motion_seq.set_element(j, i, newElement);
        }
    }
```

```
else{
    for (int j = 0; j < original_motion_sequence_.get()->spatial_size(); j++){
        double low = (i - NUM) * (original_motion_sequence_.get()->temporal_size() - 160) / (original_motion_sequence_.get()->temporal_size() - NUM) + NUM;
        if (low >= original_motion_sequence_.get()->temporal_size())
            low = original_motion_sequence_.get()->temporal_size() - 1;
        double high = low + 1;
        if (high >= original_motion_sequence_.get()->temporal_size())
            high = original_motion_sequence_.get()->temporal_size() - 1;
        //std::cout << "i: " << i << ",low: " << low << ",high: " << high << std::endl;
        //std::cout << original_motion_sequence_.get()->element(j, low) << "\n//////////\n";
        //std::cout << original_motion_sequence_.get()->element(j, high) << "\n//////////\n";
        double rxLow = math::ToRadian(original_motion_sequence_.get()->element(j, low)[0]);
        double ryLow = math::ToRadian(original_motion_sequence_.get()->element(j, low)[1]);
        double rzLow = math::ToRadian(original_motion_sequence_.get()->element(j, low)[2]);
        double txLow = original_motion_sequence_.get()->element(j, low)[3];
        double tyLow = original_motion_sequence_.get()->element(j, low)[4];
        double tzLow = original_motion_sequence_.get()->element(j, low)[5];
        double rxHigh = math::ToRadian(original_motion_sequence_.get()->element(j, high)[0]);
        double ryHigh = math::ToRadian(original_motion_sequence_.get()->element(j, high)[1]);
        double rzHigh = math::ToRadian(original_motion_sequence_.get()->element(j, high)[2]);
        double txHigh = original_motion_sequence_.get()->element(j, high)[3];
        double tyHigh = original_motion_sequence_.get()->element(j, high)[4];
        double tzHigh = original_motion_sequence_.get()->element(j, high)[5];
        math::RotMat3d_t mat1;
        mat1 = math::ComputeRotMatXyz(rxLow, ryLow, rzLow);
        Eigen::Quaterniond qLow(mat1);
        math::RotMat3d_t mat2;
        mat2 = math::ComputeRotMatXyz(rxHigh, ryHigh, rzHigh);
        Eigen::Quaterniond qHigh(mat2);

        Eigen::Quaterniond middle = qLow.slerp(0, qHigh);
        math::Vector3d_t middleAxisRadian = math::ComputeEulerAngleXyz(middle.toRotationMatrix());
        math::Vector3d_t middleAxis(math::ToDegree(middleAxisRadian[0]), math::ToDegree(middleAxisRadian[1]), math::ToDegree(middleAxisRadian[2]));
        math::Vector3d_t middlePoint((txLow + txHigh) / 2, (tyLow + tyHigh) / 2, (tzLow + tzHigh) / 2);
        math::Vector6d_t newElement(middleAxis, middlePoint);
        new_motion_seq.set_element(j, i, newElement);
    }
}
```

• Result & Discussion

- Problems Encountered:
  1. 一開始由於路徑有中文的關係，因此造成執行時會出現 Error；之後有陸陸續續又很多人向我問過著個問題，因此可見很多人都有遇到因路徑有中文而造成程式中斷的問題。
  2. 在開啟 AMC File 與 ASF File 之後，我發現兩個檔案中 bone name 的順序不一樣，然而在第一部分的 Function Input (joint_spatial_pos) vector 中，其實已經將 AMC File 的 bone 重新排列過了，因此我們所得到的 index 在兩個讀取 ASF 與 AMC 的陣列中是屬於一樣的 bone。
  3. 在讀取 Euler Angle (Rx, Ry, Rz) 時，我先將他們讀進來轉換成 rotation matrix，然而一開始使用網路上查到的 Matrix3d mat = AngleAxis(x, Vector::UnitX) * AngleAxis(y, Vector::UnitY) * AngleAxis(z, Vector::UnitZ) ，造成矩陣一直與 helper function 的 rotation 不一樣，後來改用 math::ComputeRotMatXyz(x, y, z) 就解決了。
  4. 讀取進來的 rx, ry, rz 是 degree，而必須轉成 radian 在做運算，因此也造成我一開始一直算錯的問題。
- Rotation Matrix:

  除了上述讀入的問題，我們這次作業也學到的 Rotation Matrix, Euler Angle 以及 Quaternion 的轉換，由於上課時老師說過如果要做 Slerp，則 Rotation Matrix 會出問題，因此必須轉成 Quaternion 才能做運算，做完 Interpolation 之後再做 Time Warping。

- Forward Kinematics:

  在做 FK 的時候，由於我們從 Root 開始 Traverse 整個 Skeleton，透過一步步轉換 Local Coordinate，我們可以得到正確的 Bone start position, end position 以及 rotation。若中間有一步的 rotation 算錯，則勢必會影響到後面的 bone state。

Conclusion

- 由於在做 Time Warping 時要做 Slerp，因此必須將 Rotation Matrix 轉成 Quaternion。
- 在選擇 Time Frame 時，由於是前半部動作要變慢，後半部動作要變快，因此改變的 Time Frame 要在 Target Frame 後面。
- 在做 Forward Kinematic 時，要從 root position 開始，traverse 各個分支，計算 start position，並用 local coordinate 求出來的 rotation 算出 end position，在將 end position 當作下一個 bone 的 start position，繼續完成模擬 skeleton。