Introduction to Computer Animation HW2 Report

資工系 0716234 蕭彧

- ## Introduction/Motivation

What we want to implement this time is to show Forward Kinematics and Time Warping by simulating character movement in punching and move around. In the operation, we will achieve the conversion and operation of rotation matrix and Quaternion through the existing Function and various calculations provided by Eigen. We will also calculate the start position, end position and rotation between the bones from the root of the skeleton and simulate the entire human shape.

- ## Fundamentals
  - Local Coordinate:

    There is a coordinate for each bone. We set the bone's start position as the origin, and multiply the bone's Ramc and Rasf by the parent's rotation to get the rotation at the local coordinate. Using this rotation to multiply the local direction and add the start position to get the end position. And so on, starting from root we can traverse the entire skeleton.

  - Global Coordinate:

    The world coordinate system is a coordinate system with (0, 0, 0) as the origin. Through this coordinate system, we can get the position of each point, line, and surface in space. However, when simulating the skeleton in the job, many conversions are required to calculate with global coordinates, which is inconvenient, so we use local coordinates. From the local coordinate of each bone, we can calculate the end position from the start position and rotation.

  - Time Warping:

    Time Warping is a scaling behavior for time, allowing us to change the time of different frames in the animation, such as the form of time extension or contraction. In the second part of the homework, since each frame takes the same time, there is no difference between the speed and the slowness of the action. If we use time wrapping, the skeleton can punch faster or slower depends on the target frame we align.

  - Forward Kinematics:

    Forward Kinematics means that a joint in a bone in the skeleton generates transition and rotation at a specified time, and the bone that extends after it is affected by

its rotation. That is, when the parent node moves or rotates, how child nodes are affected by it. We can find the rotation of each time point from the Root through AMC File and ASF File, and extend to the global coordinate whose end position is the start position to find the position of each bone of the skeleton.

- ## Implementation
  - Forward Kinematics:

    In this homework, I use BFS to traverse the skeleton, and implement the following formula to calculate bones start position, end position, rotation in global coordinate.

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

```
Eigen::Quaternionf rt = u->rotationParentCurrent * posture.rotations[u->idx]
```

Multiply the child to parent rotation in asf file with rotation in amc file to get global child to parent rotation.

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

```
for (Bone* itr = u->parent; itr != nullptr; itr = itr->parent) {
    rt = itr->rotationParentCurrent * posture.rotations[itr->idx] * rt;
}
```

Use for loop to calculate the quaternion from local coordinate to global coordinate.

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

```
Eigen::Vector3f dir = u->direction * u->length;
```

Since the direction in asf file is unit vector, we need to multiply the length to get correct direction.

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

```
u->endPosition = rt.toRotationMatrix() * (dir + posture.translations[u->idx]) + u->startPositio
```

By multiplying Vi plus translation in amc file by rotation plus the start position of each bone we can get the end position of that bone. and continue go down the traverse and calculate the data of the bone whose end position is the start position.

```
bone->startPosition = posture.translations[bone->idx];
bone->endPosition = posture.translations[bone->idx];
std::unordered_map<int, bool> visited;
visited[bone->idx] = true;
std::queue<Bone*> q;
for (Bone* itr = bone->child; itr != nullptr; itr = itr->sibling) {
    visited[itr->idx] = true;
    q.push(itr);
}

while (!q.empty()) {
  Bone* u = q.front();
  u->startPosition = u->parent->endPosition;
  Eigen::Quaternionf rt = u->rotationParentCurrent * posture.rotations[u->idx];
  for (Bone* itr = u->parent; itr != nullptr; itr = itr->parent) {
    rt = itr->rotationParentCurrent * posture.rotations[itr->idx] * rt;
  }

  Eigen::Vector3f dir = u->direction * u->length;
  u->endPosition = rt.toRotationMatrix() * (dir + posture.translations[u->idx]) + u->startPosition;
  u->rotation = rt;

  for (Bone* itr = u->child; itr != nullptr; itr = itr->sibling) {
    if (visited.find(itr->idx) == visited.end()) {
      visited[itr->idx] = true;
      q.push(itr);
    }
  }
  q.pop();
}
```

In the left is my whole code, first I do some preprocessing to cope with root which has same start point and end point, and then starting BFS to iterate through the bones and update their position and rotation.

- Time Warping:

  In the motion warp, I let the median frame of the old frame align to the frame 180 of the new frame (before the median frame), so the punch speed will be faster at the beginning and the skeleton's motion will slow down after he punched. For translation, I use simple interpolation formula to calculate, and for rotation I use the build-in function of quaternion, slerp, to calculate. The tricky thing is that to avoid segmentation fault, I only calculate to the final frame – 1. Let the start and final frames are same in old and new motion.

```cpp
Motion motionWarp(const Motion& motion, int oldKeyframe, int newKeyframe) {
  Motion newMotion = motion;
  int totalFrames = static_cast<int>(motion.size());
  int totalBones = static_cast<int>(motion.posture(0).rotations.size());
  int halfFrames = totalFrames / 2;
  int targetFrames = 180;
  for (int i = 0; i < totalFrames; ++i) {
    // Maybe set some per-Frame variables here
    if (i < targetFrames) {
      int low = halfFrames * i / targetFrames;
      int high = low + 1;
      double ratio = static_cast<double>(halfFrames * i / targetFrames) - low;
      for (int j = 0; j < totalBones; ++j) {
        // TODO (Time warping)
        // original: |--------------|--------------|
        // new     : |------------------|----------|
        // OR
        // original: |--------------|--------------|
        // new     : |----------|-------------------|
        // You should set these variables:
        //     newMotion.posture(i).translations[j] = Eigen::Vector3f::Zero();
        //     newMotion.posture(i).rotations[j] = Eigen::Quaternionf::Identity();
        // The sample above just set to initial state
        // Hint:
        //   1. Your should scale the frames before and after key frames.
        //   2. You can use linear interpolation with translations.
        //   3. You should use spherical linear interpolation for rotations.

        newMotion.posture(i).translations[j] =
            ratio * motion.posture(high).translations[j] + (1 - ratio) * motion.posture(low).translations[j];
        newMotion.posture(i).rotations[j] =
            motion.posture(low).rotations[j].slerp(ratio, motion.posture(high).rotations[j]);
      }
    }
    else if (i < totalFrames - 1) {
      int low = (totalFrames - halfFrames) * (i - targetFrames) / (totalFrames - targetFrames) + halfFrames;
      int high = low + 1;
      double ratio =
          static_cast<double>((totalFrames - halfFrames) * (i - targetFrames) / (totalFrames - targetFrames)) +
          halfFrames - low;
      for (int j = 0; j < totalBones; ++j) {
        newMotion.posture(i).translations[j] =
            ratio * motion.posture(high).translations[j] + (1 - ratio) * motion.posture(low).translations[j];
        newMotion.posture(i).rotations[j] =
            motion.posture(low).rotations[j].slerp(ratio, motion.posture(high).rotations[j]);
      }
    }
  }
  return newMotion;
}
```

- **Result & Discussion**
  - Problems Encountered:
    1. The biggest problem is understanding physics formula. I first don't know I need to use the rotation in amc file which cause my skeleton look very strange.
    2. The root point and its child are actually in the same position, so I need to preprocess the root start position and end position otherwise the BFS process may fail.
    3. When using quaternion, I first use the formula in the slide to convert the vector v to [0, v] and rotate the vector using quaternion. But its hard to convert it back to 3Dvector so I end up convert the quaternion to rotation matrix and just multiply it with vector v.
    4. As the above section mentioned, I first calculate through the final frame and my program keep saying I have errors, and then I figure out the high variable in my code will access the index which is beyond the vector.

  - Forward Kinematics:

    When doing FK, since we start from Root to Traverse the entire Skeleton, by converting Local Coordinate step by step, we can get the correct Bone start position, end position and rotation. If there is a wrong calculation of rotation in the middle, it will inevitably affect the following bone state.

- **Conclusion**
  - When selecting Time Frame, since the first half of the action should be faster and the second half of the action should be slower, the changed Time Frame should be in front of the Target Frame.
  - When doing Forward Kinematic, start from the root position, traverse each branch, calculate the start position, and use the rotation obtained from the local coordinate to calculate the end position, and use the end position as the start position of the next bone, and continue to simulate the skeleton.