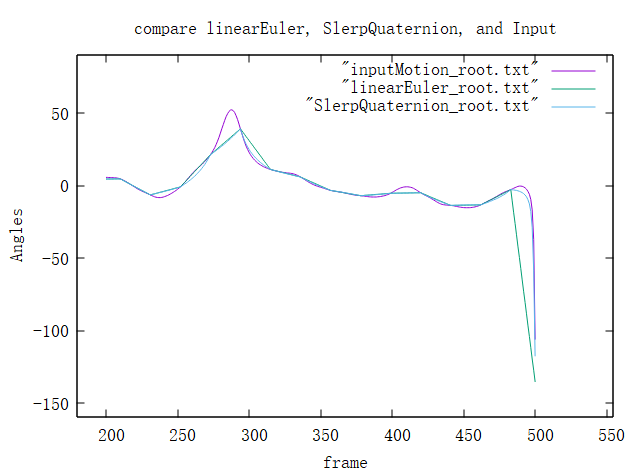
CSCI520 Hw2 Report

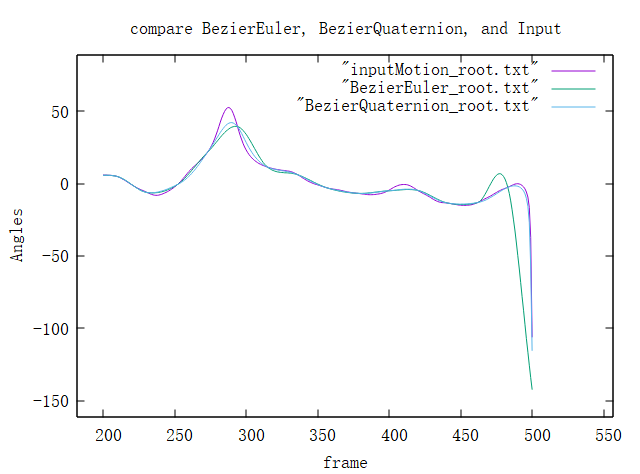
Yukun Zhou

**Findings and Observations**

For basic requirements, I implemented Bezier Euler, Slerp Quaternion, and Bezier Quaternion interpolation algorithms, and compared them with the Linear Euler method and input motion itself.

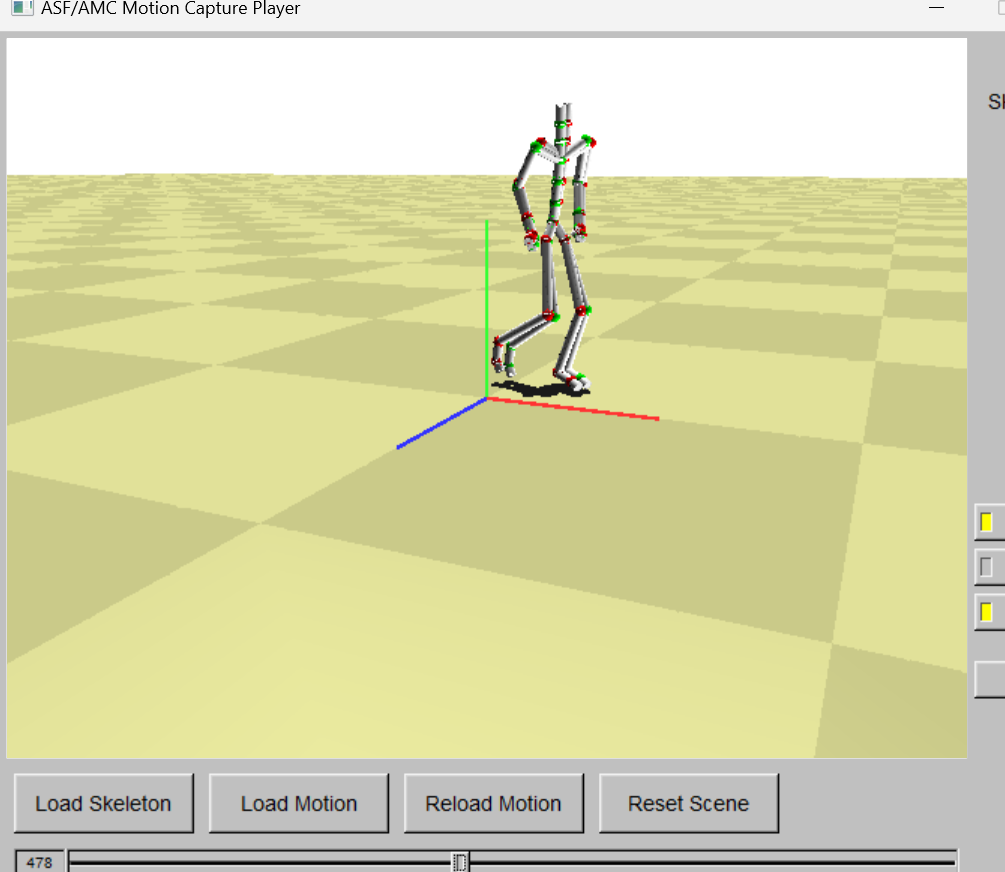
First, I will compare and discuss these algorithms by required 4 graphs. They are under



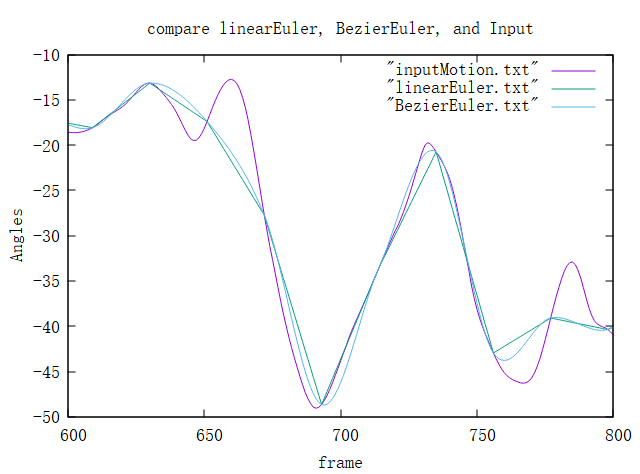
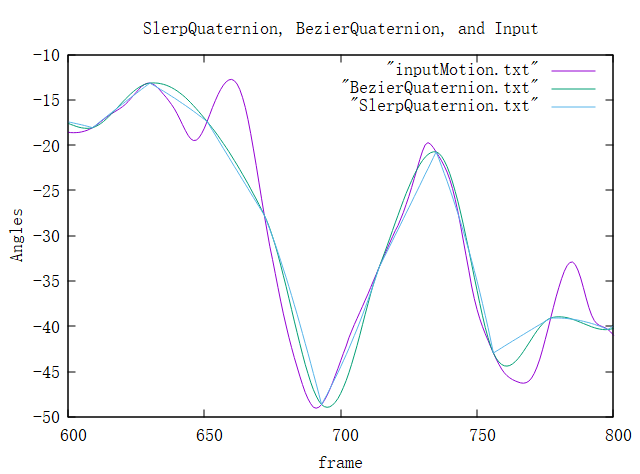


By comparing linear Euler and Slerp Quaternion, we can find that for both linear interpolations, the one with Slerp and Quaternion achieves a better result. Its angles change more smoothly and closer to the original motions but still get some stiff rotations. Because Slerp itself on Quaternion will create non-equal spaced interval on a unit sphere. We can find the Euler angle method can easily cause large mismatches between generated and original ones, and it quite depends on the keyframe selected. Only if the keyframe is just at the peak or valley of the angle curve, Linear Euler can get a better result, which is less likely to happen.

Looking into the two Bezier methods, their differences are not so obvious, and both are relatively more matched with the original motions than linear methods. However, the Bezier Euler method still shows some mismatches, and it’s quite bad around frames 450-500.

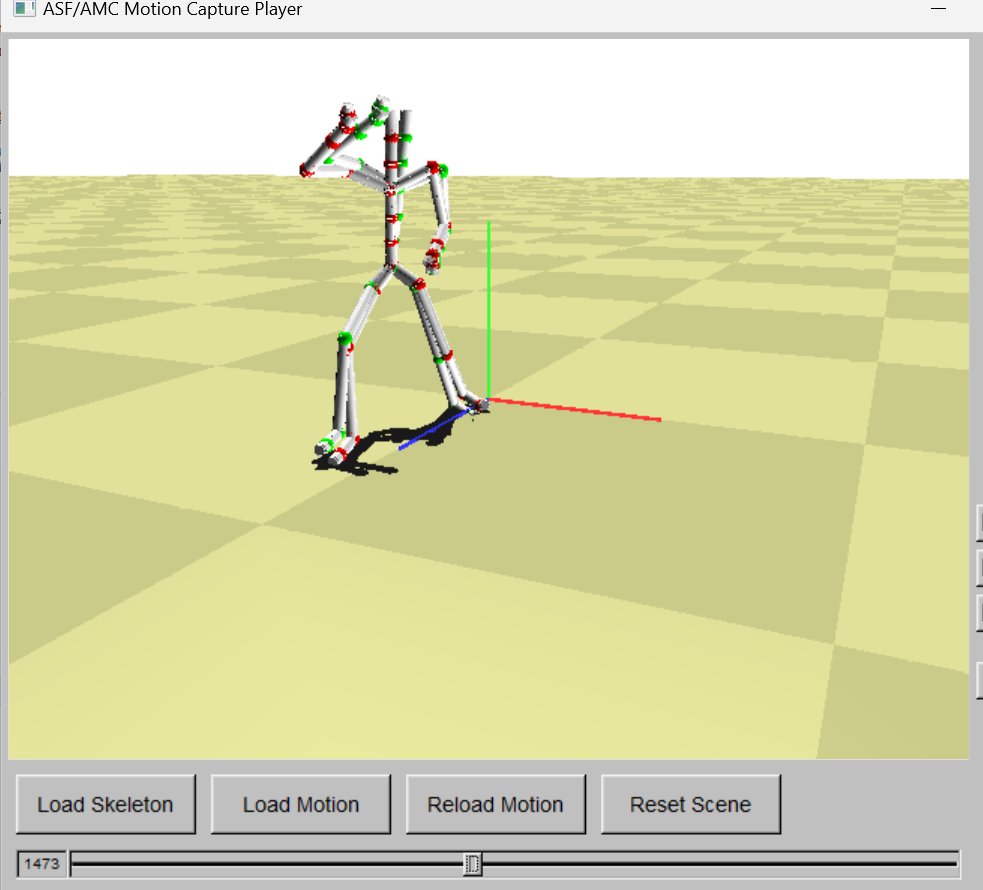
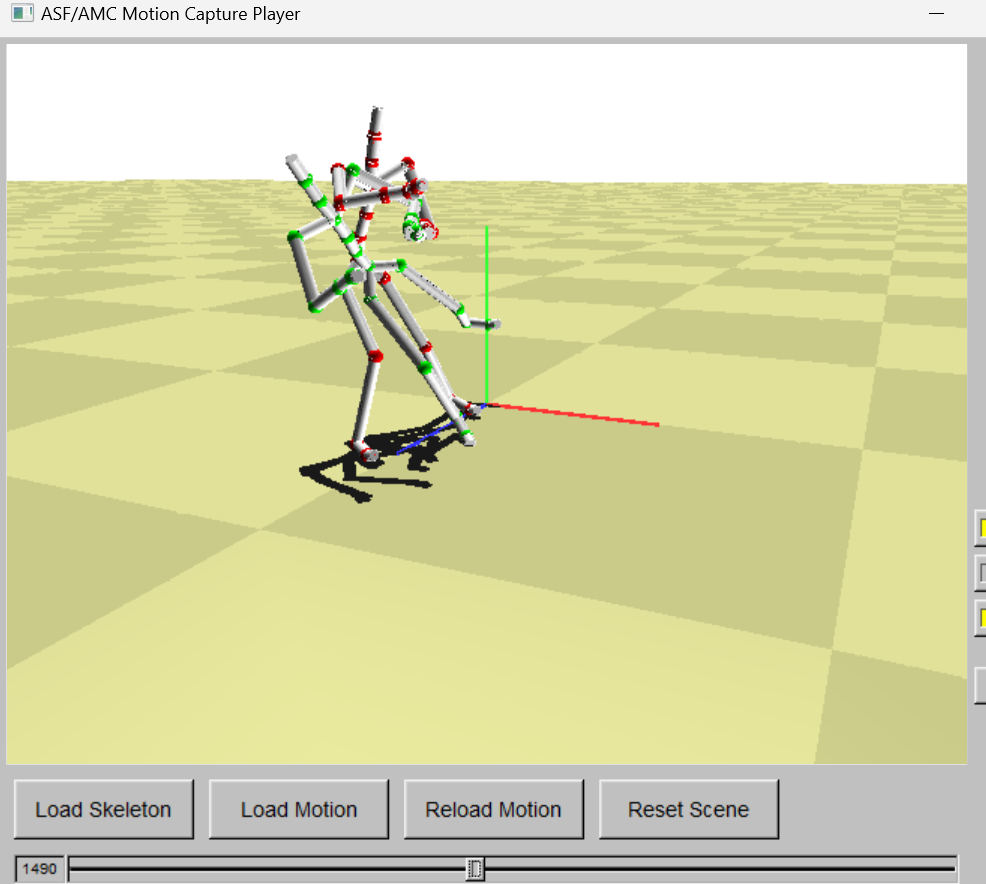
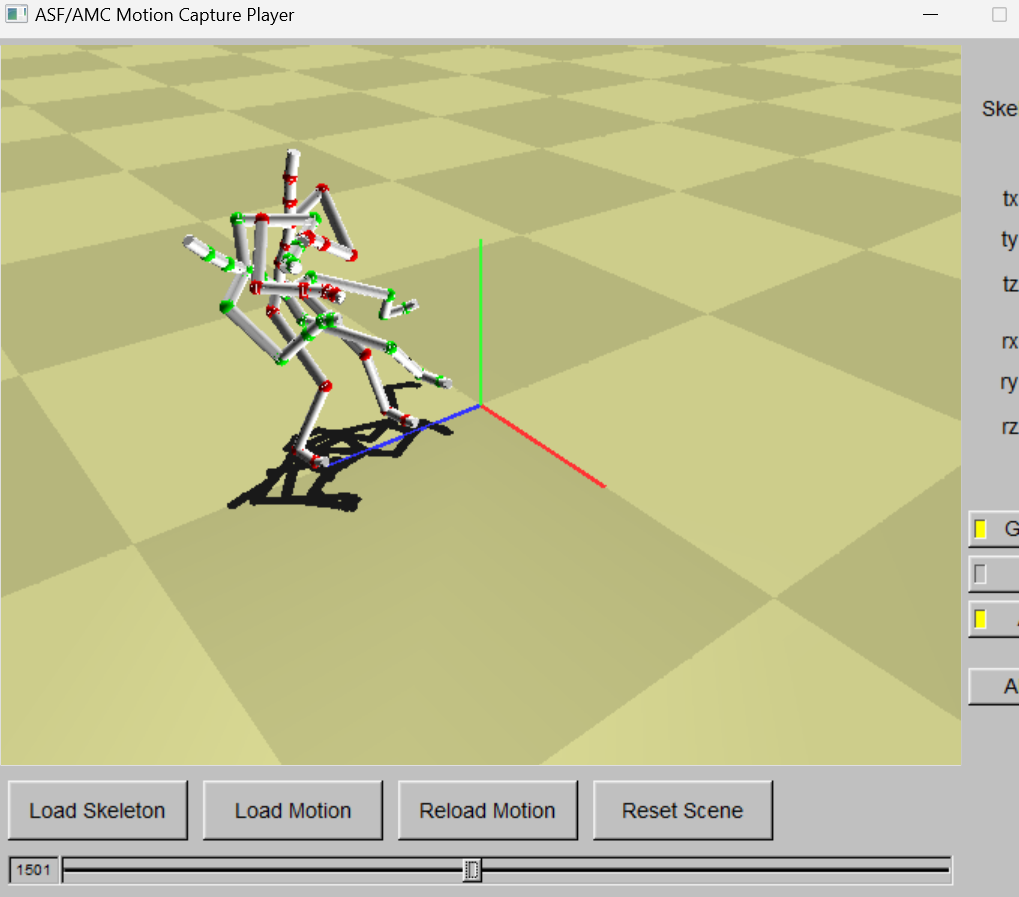
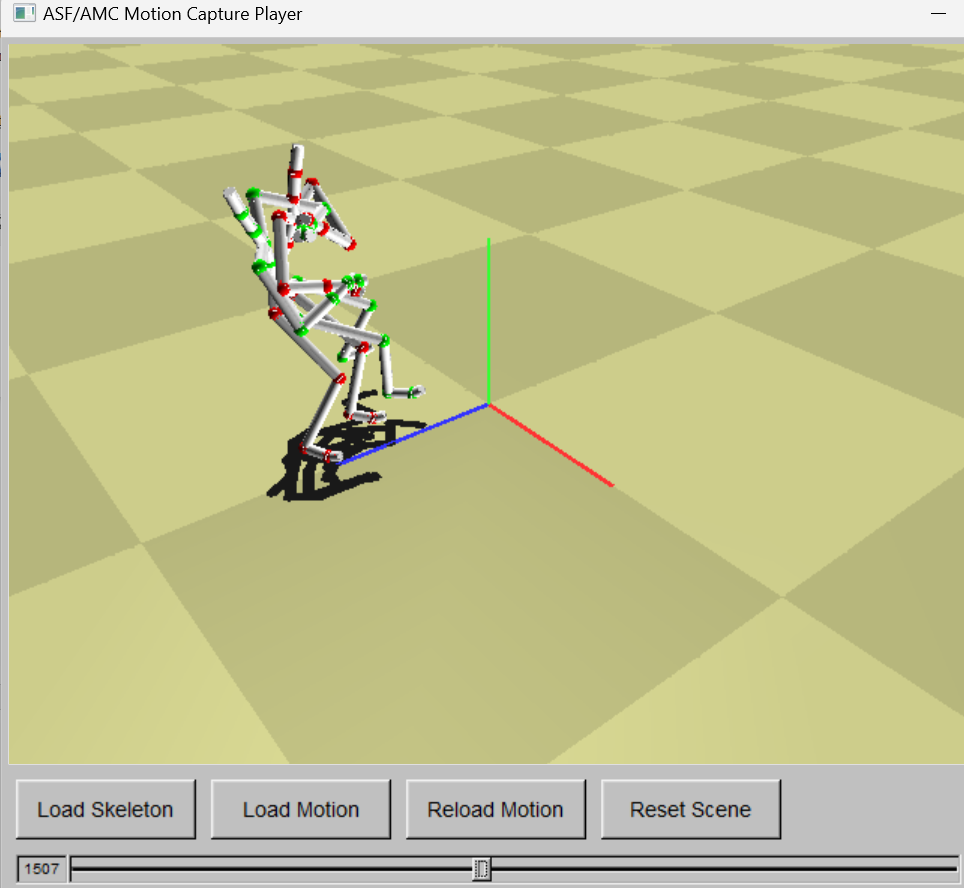


In those frames (fig above), the skeleton is doing some big rotations. Euler methods execute them at a constant speed, and Bezier Euler just distorts its angle change curve in its defined way, which are less likely than real human motion.



Our algorithm cannot know the changes with various changing directions within two keyframes. In both two graphs, the ones with Bezier splines match the original motion curve more. It may be because the Bezier splines distort the angle change in a non-constant manner, which is more likely to humans. Because human motions/rotations will have a slow speed when is near the start or the end of an action.

For three videos, they are under IDE-starter/VS2017/mocapPlayer and have corresponding names, “be, lq, bq”. we can also find Linear Euler and Bezier Euler have the worst results. We can find some quite strange rotations when the skeleton does some large rotations, which may be due to the gimbal lock. Some parts of the body cannot rotate in this scenario. Such as followed screenshot:

Those two with quaternion match the original motion much better and have no this gimbal lock problem. Some mismatches still happen, but they are not so obvious and severe like the strange motion mentioned above. Moreover, with a larger N, the performances of all four algorithms are not so good as expected.

Linear Euler is the easiest one to code and has the least execution time (about running time, I do it in the extra credit part). Bezier Euler works better than Linear Euler, and it’s more similar to the original motion. It’s kind of more time-consuming. Quaternion methods cost more time but overcome the gimbal lock problems and therefore work better in situations with large and complicated rotations. Bezier quaternion works better and costs more time than the Slerp one.

**Extra Credit**

1. **Computation Time**

I set clockers to handle this task. Here I use MaterialArt motion as examples to execute four algorithms (unit is in ms):

As I mentioned before, Linear Euler is quite fast to execute. Bezier splines methods increase lots of running time compared to just linear methods for both Euler and Quaternion methods. We need to compute many extra control points and do more computations when using DeCasteljau Construction. The quaternion method itself is slower than Euler's, perhaps because the Slerp operation is much slower than Lerp.

1. **Handle non-uniform keyframes in time**

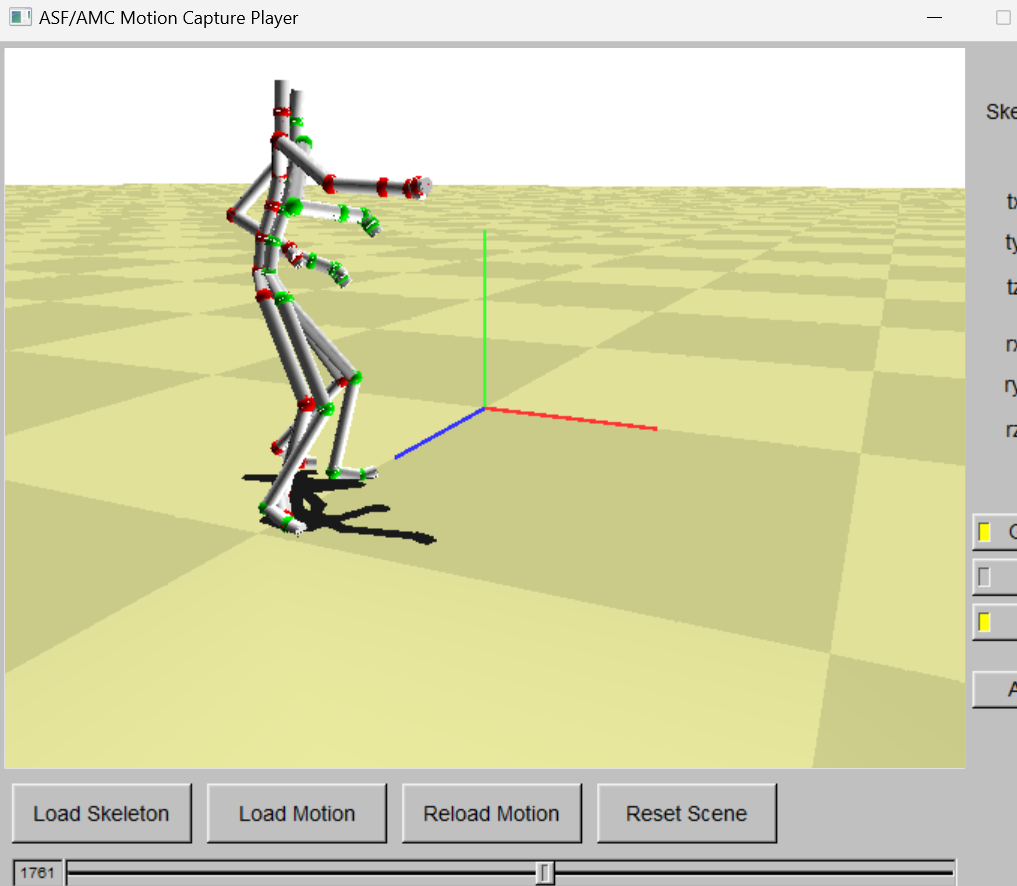
I suppose we have an extra file called “Nlists.txt” to state non-uniform keyframe gaps in time.

**One situation** is that we have the list that includes all keyframes in motion (in a non-uniform time manner). **The other situation** is that the list gives a pattern of keyframe time gap N to follow, which means we may repeat this pattern many times until the motion is ended.

To handle both, I designed a new interpolation algorithm called “NonUniform\_BezierInterpolationQuaternion” in Interpolator.cpp. We can change to this algorithm when input “n q” as the InterpolationTypeString and AngleTypeString. I also change corresponding codes in other files to make this work well. I choose BezierQuaternion because it’s the hardest to implement (quite easy to change to other algorithms with simple code deletion) and has the best performance.

The hard and tricky part of this is to consider the previous N and next N to compute a\_n and b\_nplus1 because it’s not uniform as I implemented before. I read the file to get the whole list first and make a judgment before each time’s computation. The judgment guarantees if the list provides a pattern, it can be iteratively used.

The result is also recorded as a **video** in a folder called “non-uniform”.



It has various performances, which depends on what the keyframe time gap N is at that time.

1. I changed the ground appearance.