Snapshot Interpolation

https://github.com/geckosio/snapshot-interpolation

JavaScript/TypeScript library that interpolates between frames sent over large and uneven intervals, or with great latency. By keeping a buffer of the last 3 server frames, the library animates between jumps in the server frames.

This code is specifically designed for real-time games, where accuracy is less important, but performance and visual clarity/quality are the factors being maximized.

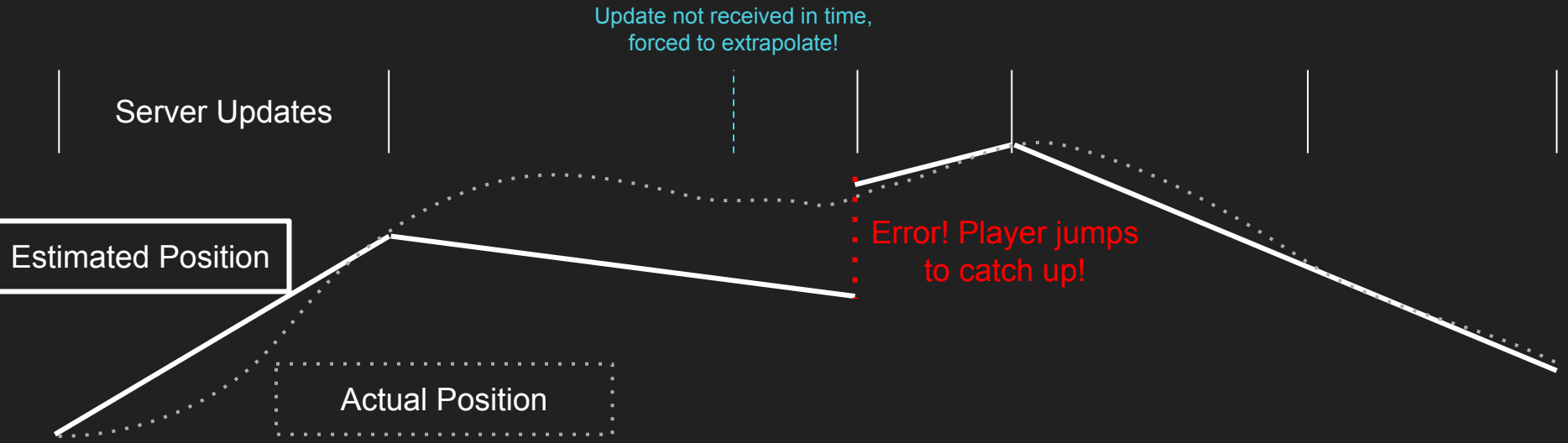
Built on the geckos.io backend engine, this interpolation library has additional features for smoother performance, especially on worse connections.
The position of server states is delayed on the client-side, often by 100ms, and it is displayed as a position between two known points in the snapshot buffer.

Geckos.io is a backend library for real-time client/server communication, written in Node.js, designed for HTML5 online multiplayer games. It is open source, and created and maintained for free.
The snapshot interpolation code is written as an extension of geckos.io, by the same developers.

Because this is designed for use in real-time games, it needs to be efficient enough to run multiple separate interpolations in significantly under 16ms. Therefore, many of the more accurate methods may be unusable for this problem, and faster estimators should be used instead.

The library currently uses linear interpolation and spherical linear interpolation, only used between two snapshots at a time. It seems like this could be modified to use other interpolation equations, and it could take advantage of the multiple frames in the buffer to create a better and smoother interpolation between the points in the data set. When stress-testing the software, I found that with a very slow frequency of server updates, there is a very large visible lag, and the players begin to "jump," which implies that linear interpolation is incomplete for solving this problem.
Because the methods of interpolation used in class are optimized for accuracy, they may not be the best fit for this problem; the interpolation system relies on the player not noticing small delays, or slightly incorrect movement, as long as it is close. It seems like modeling curves around the known points (perhaps Bezier curves) could create a smoother output. I would also experiment with using these same curves to extrapolate the output slightly, if the frames are taking too long to arrive.

Interpolation performance should be measured according to multiple metrics. Error could be summed for simulations of test data, but it is not necessarily a good measure of what would be noticeable to a player (especially if user cannot see the "true" value, it is not directly visible when the client value is wrong). It may be useful to also measure the cumulative distance of each time the player "jumps" to handle being told the client has estimated the player position incorrectly.



It seems like there could be an advantage in modifying this software to include different methods of interpolation and extrapolation, and to test which methods result in a better user experience.

In order to compare this, the axis could be "ms of latency" vs "cumulative jump distance", or vs "cumulative error." All of which would be run on a predetermined dataset.
It would also be useful to plot the "buffer size" vs "time taken to interpolate (ms)."