Downsampling Discrete Trajectories

Pascal Leroy (phl)

2026-01-26

This document describes the computations that are performed by the method `Append` of class `DiscreteTrajectorySegment` to downsample the points produced by an integrator and produce a compact yet accurate representation of discrete trajectories.

Overview

An integrator produces a stream of tuples (t_i, q_i, p_i) giving the degrees of freedom of a massless body at discrete times t_i. The purpose of downsampling is twofold:

- Construct cubic Hermite splines that interpolate between consecutive t_i to make it possible to evaluate the degrees of freedom at any time with sufficient accuracy.
- Make the representation more compact by dropping the tuples (t_i, q_i, p_i) for $i \in \left]1, n\right[$ if the Hermite spline constructed based on (t_1, q_1, p_1) and (t_n, q_n, p_n) approximates these tuples with sufficient accuracy.

A Brute Force Algorithm

The best way to describe the problem we want to solve is to present a brute-force algorithm that provides an exact solution:

Algorithm 1: BruteForceDownsampling.

Input: A tolerance η and a stream of tuples (t_i, q_i, p_i).
Output: A stream of intervals $[j_1, j_2]$ such that the Hermite spline based on the tuples $(t_{j_1}, q_{j_1}, p_{j_1})$ and $(t_{j_2}, q_{j_2}, p_{j_2})$ approximates the input tuples (t_i, q_i, p_i) for $i \in [j_1, j_2]$ with a tolerance better that η.

Let $j \leftarrow 1$.
while *not at end of input stream* **do**
    Compute the Hermite spline h based on the tuples (t_j, q_j, p_j) and (t_i, q_i, p_i).
    if $\sum_{k=j}^{i} \left\| h(t_k) - q_k \right\|_2 > \eta$ **then**
        Emit the interval $[j, i-1]$.
        Let $j \leftarrow i - 1$.
end
Emit the interval $[j, n]$ where n is the index of the last element in the stream.

A few things are worth noting here:

- Only the tuples $(t_{j_1}, q_{j_1}, p_{j_1})$ and $(t_{j_2}, q_{j_2}, p_{j_2})$ corresponding to the bounds of the intervals $[j_1, j_2]$ need to be stored permanently. The other tuples can be interpolated with an accuracy better than η.
- The Hermite splines do not need to be stored permanently; they can be reconstructed based on the tuples $(t_{j_1}, q_{j_1}, p_{j_1})$ and $(t_{j_2}, q_{j_2}, p_{j_2})$.
- The algorithm is optimal in the sense that it produces the longest possible intervals $[j_1, j_2]$ that satisfy the tolerance η.

- For each input tuple, the algorithm needs to scan all past tuples since the upper bound j of the last emitted interval. Therefore, the algorithm is quadratic in the number of tuples in the input stream.

A Binary Search Algorithm

In order to avoid the quadratic complexity, we used to use a binary search algorithm as follows:

Algorithm 2: BinarySearchDownsampling.

Input: A tolerance η, an integer N, and a stream of tuples (t_i, q_i, p_i).
Output: A stream of intervals $[j_1, j_2]$ such that the Hermite spline based on the tuples $(t_{j_1}, q_{j_1}, p_{j_1})$ and $(t_{j_2}, q_{j_2}, p_{j_2})$ approximates the input tuples (t_i, q_i, p_i) for $i \in [j_1, j_2]$ with a tolerance better that η.

Let A be an array of size N used to store tuples.
while *not at end of input stream* **do**
    if A *is full* **then**
        Let $k \leftarrow \lfloor N/2 \rfloor$.
        Emit `SearchInterval`(A, 1, k).
        Emit `SearchInterval`(A, $k+1$, N).
        Clear A.
    Append (t_i, q_i, p_i) to A.
end

Function `SearchInterval`(A, i_1, i_2) **is**
    Compute the Hermite spline h based on the tuples $(t_{i_1}, q_{i_1}, p_{i_1})$ and $(t_{i_2}, q_{i_2}, p_{i_2})$.
    if $\sum_{k=i_1}^{i_2} \left\| h(t_k) - q_k \right\|_2 > \eta$ **then**
        Let $k \leftarrow \lfloor i_2/2 \rfloor$.
        Emit `SearchInterval`(A, i_1, k).
        Emit `SearchInterval`(A, $k+1$, i_2).
    else
        Emit $[i_1, i_2]$.
    end
end

This algorithm has the following properties:

- The array A must be stored permanently since it contains tuples that have not been downsampled yet and that will be needed to make a future decision about the intervals to emit.
- The choice of N involves a trade-off: N must be large enough that the downsampling is effective and produces long intervals $[j_1, j_2]$; but it must not be so large that the size of A affects performance.
- The algorithm has a complexity of $O(N \log N)$ on average and $O(N^2)$ in the worst case.
- The algorithm does not produce the longest possible intervals: in the worst case, all the emitted intervals may be too small by a factor of 2.