Curve Extrapolation and Visualization Task

Ashlesh Pandey

March 2024

1 Problem Statement

Develop a Flutter application with the objective of enabling users to manipulate the positions of five draggable points labeled as 1, 2, 3, 4, and 5 on the screen. The application should generate a smooth curve connecting these five points seamlessly. Additionally, the application should feature a ball with a diameter of 20 pixels on this newly generated screen. The task involves programming the application to extrapolate the movement of this ball along the drawn curve.

2 Task Breakdown

1. **Set Up Flutter Project:**
   - Create a new Flutter project.
   - Set up necessary dependencies for drag-and-reposition, handling gestures on custom painters using touchable.
2. **Implement Drag-and-Reposition Functionality:**
   - Create draggable custom paint objects for each point (1, 2, 3, 4, and 5).
   - Implement logic to handle dragging and repositioning of these widgets.
3. **Draw Smooth Curve:**
   - Utilize cubic Bézier splines to create a smooth curve that connects the five points.
   - Implement algorithms or libraries that calculate the control points for the cubic Bézier curves based on the positions of the draggable points.
4. **Extrapolate Proof Ball Movement:**
   - Calculate the points on the curve using the PathMetric of the path.
   - Implement logic to extrapolate the movement of the proof ball along the drawn curve with a Tween Animation from 0 to 1, representing the progress of the ball along the drawn curve.
   - Track the positions of the proof ball for fitting them on the curve.
   - Develop logic to calculate non-overlapping positions for the balls along the drawn curve.

3 Implementation

Repositioning Nodes

- A basic Gesture Detector was sufficient to update the Offset's of individual nodes.
- The package touchable's implementation along with a callback function returning the index of the currently selected node was necessary since Custom Paint objects can't receive gesture inputs for interaction.

Spline Interpolation

- Preliminary research for interpolating splines pointed towards Bézier curves, so first few iterations of the code included manual control points that could be moved similar to the nodes, based on which third-order Bézier curves would be drawn across each section with the help of proste_bezier_curve and the cubic path in Custom Paint.
- Following that, the control points were calculated from the two extremities in each section.

A brief mathematical insight, the cubic Bézier polynomial in

$$B(t) = (1-t)^3 \cdot P_0 + 3(t - 2t^2 + t^3) \cdot P_1 + 3(t^2 - t^3) \cdot P_2 + t^3 \cdot P_3 \tag{1}$$

where,

- $B(t)$ represents the cubic Bézier spline function.
- P_0, P_1, P_2, P_3 are the control points.
- t is a parameter ranging from 0 to 1, indicating the position along the spline cu

Since the spline has to be continuous, the first and second derivatives to be continuous across the spline boundary.

$$B'(t) = -3(1-t)^2 \cdot P_0 + 3(1 - 4t + 3t^2) \cdot P_1 + 3(2t - 3t^2) \cdot P_2 + 3t^2 \cdot P_3 \tag{2}$$

For any i^{th} node, we have $P_{0,i} = P_{3,i-1} = K_i$, so, for any i^{th} boundary, at the left side,

$$B'_i(0) = B'_{i-1}(1) \tag{3}$$

$$-3P_{0,i} + 3P_{1,i} = -3P_{2,i-1} + 3P_{3,i-1} \tag{4}$$

$$2K_i = P_{1,i} + P_{2,i-1} \tag{5}$$

The second derivative is given by,

$$B''(t) = 6(1-t) \cdot P_0 + 3(-4 + 6t) \cdot P_1 + 3(2 - 6t) \cdot P_2 + 6t \cdot P_3 \tag{6}$$

$$-2P_{1,i} + P_{2,i} = P_{1,i-1} - 2P_{2,i-1} \tag{7}$$

At this point if we use Equations 5 and 7, there are $2(n-1)$ equations and $2n$ unknowns. Therefore, two boundary conditions, $B''(0) = 0$ and $B''_{n-1}(1) = 0$ are setup assuming that the curve becomes linear at the end points.

$$K_0 - 2P_{1,0} + P_{2,0} = 0 \tag{8}$$

$$P_{1,n-1} - 2P_{2,n-1} + K_n = 0 \tag{9}$$

Further simplification on 5 and 7 gives,

$$P_{1,i-1} + 4P_{1,i} + P_{1,i+1} = 4K_i + 2K_{i+1} \qquad i \in [1, n-2] \tag{10}$$

From Equations 8 and 9 at boundary we have,

$$2P_{1,0} + P_{1,1} = K_0 + 2K_1 \tag{11}$$

$$2P_{1,n-2} + 7P_{1,n-1} = 8K_{n-1} + K_n \tag{12}$$

This system is a tri-diagonal system for P_1, which can be solved using the tri-diagonal Algorithm. Upon obtaining P_1, P_2 can be obtained as,

$$P_{2,i} = 2K_i - P_{1,i} \qquad i \in [0, n-2] \tag{13}$$

$$P_{2,n-1} = \frac{K_n + P_{1,n-1}}{2} \tag{14}$$

Points on Curve

- The tangential offset at a position on the curve is calculated based on the Pathmetric that comes from Flutter's *computeMetric()* function on the Path.
- The points on the drawn curve were calculated based on a progress parameter ranging from 0 to 1. This comes from a Tween Animation and helps in moving a proof ball from the start to the end of the curve.
- Overlaps on the various proof balls were calculated by comparing the Eucledian distance between two interpolated points and the spacing between the balls.
- Consecutive positions of non-overlapping proof balls are calculated using the *lerp()* function with the steps (number of balls that can fit between the two points) being clamped from 0 to 1.
- Final clipping path (a semi circle with its center at the final node position) based in the direction of the normal of vector between the final node and final drawn circle's center clips the last circle if it extends outside the curve.