Impulse-style scroll animations

Matt Amert

Goal

Bring Impulse-style (i.e. EdgeHTML-style) scroll animations as an option to Chromium, behind a new feature flag.

Background

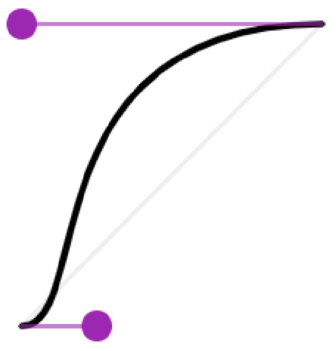
EdgeHTML has a specific style of scroll animations that give it a personality not found on other platforms. The main idea is that each “tick” of the mouse wheel tries to mimic a physical-based world where content starts moving quickly (an impulse) and then slows due to friction. One of the benefits of this approach is that the scroll feels more responsive due to the quick ramp-up at the start.
This scroll animation has already been ported into the Chromium-based Edge browser, and is enabled by default for Windows in both Dev and Canary channels.

Comparison

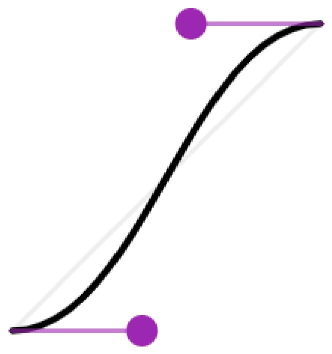
Under the hood, the implementation is very similar to that of Chromium’s scroll animation. Both are cubic Bezier curves with logic to handle velocity matching when an animation is interrupted. There are some key differences, however:

Control points

The most obvious differences are the control points for the generated cubic Bezier.



Impulse-style cubic bezier (EdgeHTML’s scroll animation curve)



Ease-in-out cubic bezier (Chromium’s scroll animation curve)

Specifically, the default curve generated for the two animation curves have the following control points:

- (EdgeHTML) Impulse-style: (0.25, 0), (0, 1)
- (Chromium) Ease-in-Out: (0.42, 0), (0.58, 1)

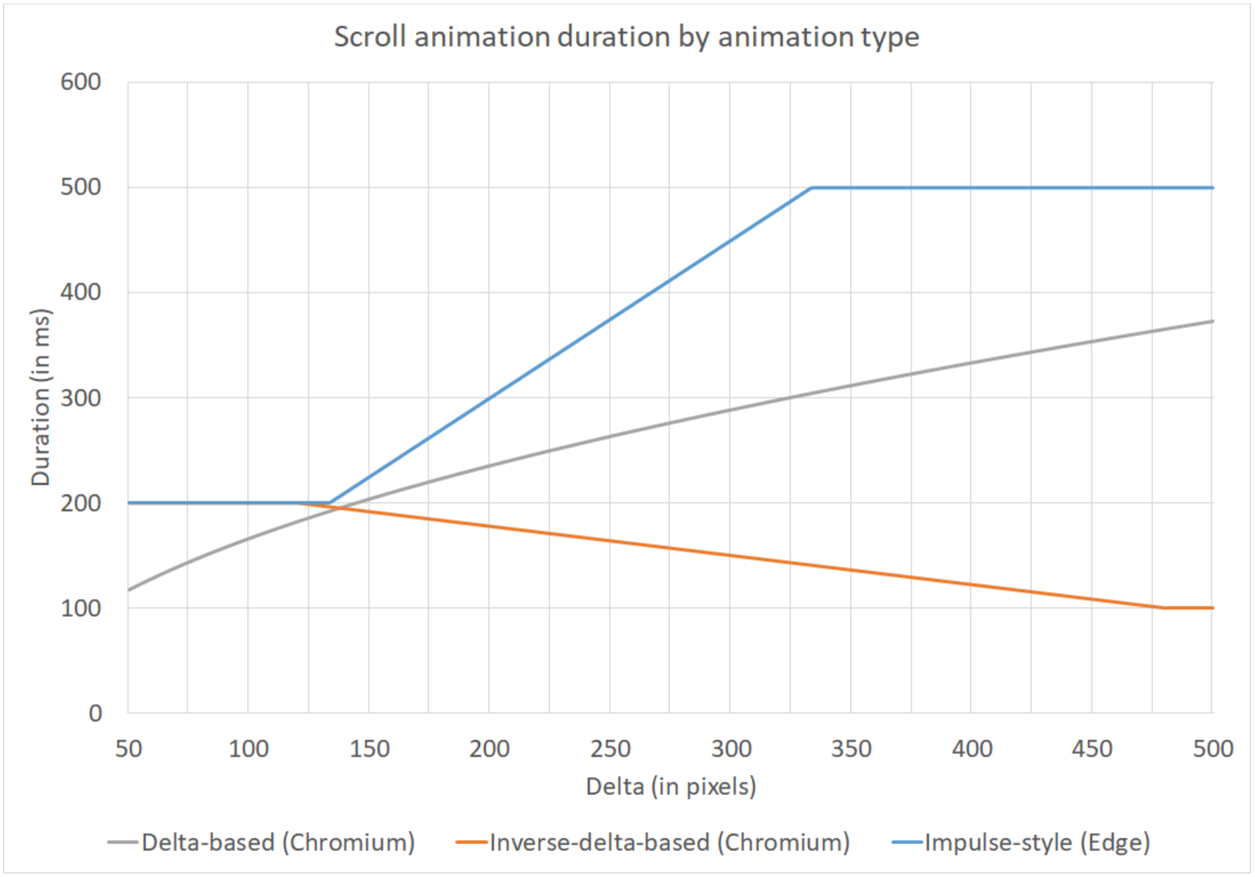
The control points used by the Impulse-style provide a sharper initial acceleration and slower decay of the velocity, as shown in the above graphs.

Duration

Impulse-style curves (used by EdgeHTML) give 1.5 milliseconds per physical pixel it needs to animate over, clamped to be in the range of 200ms and 500ms.

The Chromium-style animation curve, however, has three different modes for duration:

1. Delta-based: the duration is scaled to how far it needs to animate, used for programmatic scrolls.
2. Inverse-delta-based: the duration is scaled inversely to how far it needs to animate. This is used for mouse wheel scrolls.
3. Constant: the duration is a constant (150ms), which is used for keyboard scrolls.



Updating a running animation

When the user scrolls again while an animation is already running, both implementations attempt to velocity-match to the previous running animation. However, there are some subtle differences between the two:

1. The Chromium-style curve checks if the new intended scroll offset is close to its current target. If they are close, the animation is not updated. Impulse-style specifically removes this, since the animation still should be updated as otherwise the user input is effectively ignored.
2. Impulse-style curves set the velocity to 0 if the new target is in the opposite direction that we are animating to prevent rubber-banding. Chromium-style curves do not do this, though there is logic to prevent rubber-banding from having too big of an impact.
3. Typically velocity matching is done by scaling the y-value of the first specified control point by the velocity. However, since the velocity can be quite large for Impulse-style curves, it is possible for the scaled y-value to exceed 1; this will cause the generated bezier curve to rubber-band. Therefore, when this happens, Impulse-style curves will clamp the y-value to 1 and inversely scale the x-value to get a sharper curve.

Proposed refactoring

In order to support this new animation, we'd like to introduce some refactoring around the ScrollOffsetAnimationCurve class. Currently, consumers of this type are expected to provide the initial Bézier curve. This is only ever an Ease-in-Out curve or a Linear curve, so it should be simple enough to encapsulate the creation of these types of scroll animations with their own functions.

Additionally, for the feature, every place there is an Ease-in-Out curve should be swapped for an Impulse curve. As such, the proposed interface for creating an animation curve will be two functions:

1. CreateDefaultAnimation, which creates either an Ease-in-Out animation curve or an Impulse curve, depending on the state of the new feature flag.
2. CreateLinearAnimation, which creates a Linear animation that is used for composited scrollbar scrolling.

This will mean that we also need to refactor the DurationBehavior type into two different types:

1. DurationHint, which supplies the duration enumeration for the Ease-in-Out animation (though it is ignored by Impulse-style and Linear animations)
2. AnimationType, which denotes which type of animation the object represents (Ease-in-Out, Impulse, or Linear).

Remarks

There is a small quirk in the original implementation for EdgeHTML. Essentially, it tried to predict the time that the next rendered frame would be composited to the screen and tick its animation based on that. This had an unfortunate side-effect of skipping the first couple frames for every scroll animation, and then accidentally using values from the future when doing animation-stitching. This overall caused the animation to be more aggressive than intended.

We have decided not to port over this quirk; as such, this animation is not going to be exactly the same as the one in EdgeHTML.