Seamless Transitions

point of contact: vmpstr@chromium.org; last update: Oct 8, 2020

Motivation

The web is getting richer and more visually appealing as time progresses. New APIs that allow animations, backdrop filters, and other visual effects are being added to the web toolkit on an ongoing basis. One aspect that lacks support, however, are transitions between logical pages, which includes both navigation across same- and cross- origin domains, as well as single-page applications navigation. Specifically, it is in some cases hard, and in others impossible, to have a composed experience with smooth and seamless transitions between different logical pages. JavaScript frameworks can only go as far as providing seamless transitions for single-page applications (e.g. ramjet). In order to allow seamless transitions across different physical pages, and even different domains, browser support is needed.

This document proposes an API that enables seamless transitions between any two navigation states, whether it is a cross-origin navigation or a single-page application transition.

Terminology

- **Single-Page Application (SPA)**: SPA is an application that does not use browser navigation to transition between different application pages and states. The document remains the same as DOM is changed to give an illusion of navigation.
- **Multi-Page Application (MPA)**: MPA is an application that spans multiple pages and uses browser navigation to transition between them. In this doc, we also use this for any case where a browser navigation occurs, including cross-origin navigation to a different logical application.
- **Source Page**: The page that initiates the navigation (in both the MPA and SPA sense).
- **Destination Page**: The page that is navigated-to by the source page.
- **Root Element Transition / Root Transition**: The root element consists of all of the visual output produced by the page (i.e. the document.body). The root element transition is a transition from one page to another, whether it is an MPA or SPA navigation.
- **Shared Element Transition**: Shared element is any element other than the root element that is shared between the source and destination pages. The shared element transition is a transition from the source page's element to the destination page's element which happens independently of the root element transition.
- **Pixel output of an element:** When an element is drawn to screen using graphics hardware, there is an opportunity for the browser to keep the output for later use, which is referred to in this doc as the pixel output of an element. Note that animations that occur using pixel outputs (as opposed to the underlying DOM elements) are limited: for

example, re-rasterizing text at a different font size or color is not possible; the only possible operations act on the pixel output as a whole.

Proposal

The proposal for the new API can be split into two components based on the different desired behaviors. First, how does one transition the root document, which constitutes *all* of the visible content of the page from one navigation state to another. Second, how does one optionally specify a set of elements that are shared between the source and destination pages and that need to transition from the source page to the destination.

Root Transitions

The source page can specify one of several ways that it can navigated away from[1]:

- **Clean cut**: this is the default state where the existing source page is instantly[2] replaced by the destination page
- **Slide**: the current content slides towards a specified direction. The new content slides from the opposite direction and replaces the current content.
- **Explode**: the current content scales up and fades revealing the new content underneath.
- **Fade**: the current content fades revealing the new content underneath.

Note that this set of transitions is a sample set which can be changed depending on the developer interest.

The transition is accomplished by taking the last pixel output of the source renderer, stashing it, and then interpolating a frame from this stashed pixel output and the latest pixel output of the destination renderer. In terms of timing, this means that the animation begins at the point when the destination renderer has produced the first frame of the pixel output. In a clean cut and existing navigation experiences, this is the point at which the user would normally see the destination renderer's content.

Shared Element Transitions

The source page should also be able to specify a set of elements that need to transition to a similar set on the destination page. One of the challenges of this proposal is identifying an elegant way to identify which elements are shared. This is a difficult task since the destination page can be located in a cross-origin domain, be a part of a different application, and be the destination from multiple unrelated source domains.

Some possibilities for shared-element identification are listed below.

---

[1] These are borrowed from the Android activity transitions

[2] Instantly, but subject to network speed, rendering speed, and pre-rendering techniques.

- Id
    - The easiest identifier to use is the element id. The page can identify a specific set of ids as the elements to transition. If those ids also match the destination page, then the animation happens between the elements.
    - Pro: simple to use and is already a unique identifier of elements
    - Con: too vague, since different pages can have same ids for unrelated content, so accidental transitions can happen.
    - Con: the destination page can change ids and thus break the intended animation
    - Con: there is no way for the destination page to “opt-out” of this behavior
- Query selector.
    - Adapted from the previous efforts, <meta name="transition-elements" content=".interesting_class">. If the query selector matches the elements on both source and destination sides then they animate
    - Pro: allows both source and destination to control which elements are ‘animatable’
    - Pro: allows the destination to opt-out of the animation behavior by not specifying anything
    - Con: it is ambiguous which element on the source matches an element on a destination, since query selectors can select multiple elements.
    - Con: this is an opt-in behavior for the destination page (by default, no destination page can be animated to).
- Query selector with ids only
    - Eliminates a ‘con’ from the query selector option, by making it unambiguous which elements match to each other. In other words, this selects only ids which uniquely identify elements.
- Custom annotation
    - Custom attribute on each element that allows transition
    - Pro: fine grained control, and settings for each of the elements
    - Con: it’s unclear if the benefits of this control outweigh the extra overhead needed on top of the other options
- Other options?

Assuming the identification of shared elements can be done elegantly, the transition itself follows similar steps to the root transition. Specifically, it takes a pixel copy of source elements, performs the navigation, identifies elements that are shared on the destination page after receiving the first pixel frame of the new state, transitions **position**, **size**, and **opacity** from the source element to the destination element for each of the shared elements. Note that each of these property interpolations should be customizable (e.g. size scaling vs repeating background; ability to do a ‘clean cut’ element transition while the root transition is not ‘clean cut’).

API Proposal

https://gist.github.com/jeremyroman/0635fe159109203248ba4f95ed2db511

We introduce a new object called **PageTransition**, which controls the seamless transition for both SPA and MPA cases. The transition consists of two parts, preparation and animation:

- **Preparation**: In order for us to stash the textures required to perform the animation, it is necessary to have a preparation step in which the user agent has an opportunity to composite, annotate, and communicate needed texture to the system responsible for GPU acceleration.

  As part of this process, we would have to know which elements are going to animate. We specify this in the PageTransition constructor, along with any other properties such as the duration of the animation.

- **Animation**: Once the preparation promise has resolved, we can and should begin the animation.
  - In the SPA case, this means that we can manipulate our DOM in any way that reflects the “new page”, followed by a call to ‘start’.

    When ‘start’ is invoked, the user agent then renders and paints the new page, compositing any elements that are identified as shared in the new DOM, and communicates the annotated information to the GPU acceleration system. At this time, the GPU system performs the animation.

  - In the MPA case, we request a transition immediately. This means that the navigation occurs. During the navigation, the browser instructs the GPU acceleration system on how to identify the new renderer, and match it with the preparation results.

    Once the system receives the new annotated information, the animation is performed.

The timing of `'prepare'` and `'start'` steps is important, since script should not begin a transition before it has been prepared. Also, preparing a transition, and delaying the start arbitrarily likely has memory and behavior implications: the user agent has to retain the pixel output for an arbitrary length of time; when the transition starts, it is the saved pixel output that’s used as opposed to the current or recent pixel output. For this reason, we require that ‘start’ be called in the same task that the ‘prepare’ call’s promise was resolved.

Note that this API is a work-in-progress. We welcome any feedback that shapes the API in a way that makes it more usable.

Previous Efforts

- Intent to implement: Navigation Transitions (2014) (design doc) (tracking bug) (talk)
- Thoughts in 2015 by a Mozilla engineer
- https://schepp.dev/posts/today-the-trident-era-ends/ IE had "page transition filters"

(not publicly available):

- Design sketch
- Surface Worker