Shared Element Transition Concepts

khushalsagar@/jakearchibald@, August 2021

This is an explainer to highlight the conceptual problems we need to solve to allow a seamless transition between documents on the web. A similar capability already exists on most native platforms (Android, iOS/Mac and Windows). The feature targets two semantic transitions:

- *Single Element Transition*: An animation which specifies how an element in the previous document or document state (for same document navigation a.k.a SPAs) exits the screen or how an element in the new document enters the screen. These elements could be the root element for either document which we call a *Root Transition* (example). The elements used during a transition (including the root) are called shared elements in the rest of this explainer.

- *Paired Transition*: An animation which transitions an element that exists in both documents (example). These elements don't need to be visually identical (example).

The focus in this explainer is to highlight conceptual problems this feature needs to solve to support same origin cross-document navigations, i.e., Multi Page Apps (MPA). We expect this foundation can be used for transitions within the same document (e.g. SPA) as well.

A rough API which encapsulates ideas mentioned under "Current Proposal" in the sections below is outlined here.

TLDR; Questions

- Visual representation of the old state during the animation. The current proposal is to capture a pixel snapshot of the DOM sub-tree with the local clip, filter, opacity, clip-path etc. We preserve the transform from ancestors but no clipping (overflow or clip-path), blending, filters, backdrop-filter, etc.

- How elements from both Documents are combined : paint order, root stacking context, containing block. This also includes defining behaviour if a shared element is occluded by a non-shared element.

- How do we allow coordinating existing CSS animation APIs with animations executed by this feature. Should we introduce a new timeline or use DocumentTimeline.

- Should the new Document remain interactive during the transition? No hit-testing during transition because layout data structures don't have the correct position of elements.

- Deciding control of first paint for a new page load. Which elements should block first paint and for how long? We can consider this feature as a precursor to transitions and build on top.
    - Should the developer be able to modify the transition based on how much content loaded within a deadline?

- Syntax for specifying animations for a transition : Keyframes from WA-API, CSS transitions, etc.

Concepts

First Paint

When a user navigates from page A to B, the browser needs to decide when the tab should switch to showing content from page B. This is essentially a trade-off between a fast page load and a smooth transition:

- Prioritize displaying a frame with partial content to give quick feedback to the user, particularly when they are on a slow network.

- Wait until more content has streamed in to avoid a white flash and excessive layout shifting.

Browsers already make this tradeoff internally (Chrome's Paint Holding for example). With this feature, we'd like to enable the developer to control this experience:

- Which elements should block first paint and for how long?

- If an element is used in a paired transition and fails to load within a deadline, what should the fallback be?

Defining this behavior is also important to ensure interoperability. The alternative would be to decide first paint using heuristics which could be different across browsers.

Current Proposal

The current proposal addresses this by introducing an event that allows developers to block first paint on async activity. But it's unclear at what stage of document parsing this event is fired and ensuring an idiomatic way to register for the event before it is fired.

This problem is common to both cross-document and same document transitions, but the semantics of an event which allows asynchronously updating the DOM to the next scene are easier to reason about for same document transitions.

Document Liveness

Which state of each document the animations are executed on:

- Do we have a live version of both DOMs active during the transition? It could allow richer animations, including ones which require layout, which run independently within each document for single element transitions.

  But it doesn't help the common use-case we've seen so far: paired transitions. Most UX involves an element on the old page morphing into another. So the model needs to provide a way to set up animations which combine sub-trees/elements from two DOM trees. It also makes it difficult to reason about a universal timeline for the transition.

- If only the new document is live during the transition (or the new version of the DOM in the same-document case), what state do we capture from the old DOM? This state will be used to drive animations during the transition.

- Do we want to allow setting the animation from script, instead of a completely declarative model? This requires script in the new document to reference captured elements from the previous document.

- Should the new document's painting be paused during the transition? This makes it difficult to understand the end state for an element's animation. For example, a shared element in the new document might move after the transition starts. But doing this will cause the element to pop to the new state after the transition ends anyway.

- Should the new document remain interactive during the transition? Hit testing is bound to be incorrect since the user will see snapshots of old content that are not a part of layout data structures used for hit testing. We also don't want the new page to observe any side-effects of the animation (not a limitation for MPA but required to prevent cross-origin information sharing). So dispatching an event while the element is animating is not an option. Since the element would be at a position different from its end state.

Current Proposal

The current proposal is based on executing the transition with a live DOM for the new document. We capture independent pixel snapshots for elements in the old DOM that are animated during the transition. "Independent" here means that an element snapshot doesn’t include visual content of nested shared elements. The shared element still takes space during layout but is not painted, exposing the background it was earlier occluding. This is similar to what browsers currently do when an element is promoted to a compositor layer.

Script in the new document can refer to these snapshots (but can't access the pixels) and establish a universal timeline for the entire transition. The root element for the previous document is itself a snapshot in this model.

In addition to the snapshot, we capture metadata that’s necessary to draw each element with the same visual representation during the transition (like it’s viewport position). And to allow a seamless transition between the states in the two documents, to the extent possible with a snapshot.

We don’t pause rendering for the new document but we provide an event for the page to know when the transition finishes, so the developer knows when it’s safe to change the new DOM. If there is any change, we make a best effort to adjust the animations to allow transition to the updated state to be smooth (updating the end state for a paired transition, using the latest content for new elements when cross-fading).

We keep the Document interactive in the current implementation. But the plan is to change that to either abort the transition on any user input or provide an event to defer this decision to the developer (abort the transition or continue). The latter will only be an option for same origin transitions.

Combining Two Documents

We need to define how content from the two documents is combined during the transition. This is particularly tricky since an element’s visual state is dependent on its position in the DOM hierarchy. And a transition requires us to move elements between two hierarchies:

- A shared element could be clipped due to an overflow clip from an ancestor or a clip-path. Are these clips retained during the transition? If so, how do we animate them as this element moves to the new hierarchy during the transition? The behaviour for 3-d transforms needs to be similarly defined.

- Similar to clipping, an element inherits a lot of visual effects from its ancestors: opacity, filter, mask, transform.

- A shared element could also affect visual representation of its ancestors through backdrop-filter. Which content is this applied to during the transition?

- A shared element could be occluded by another element on the page. If we capture independent snapshots for shared elements, retaining the paint order is hard. This would require the browser to also retain independent snapshots of occluding elements, which is a fairly complicated algorithm for compositing in browsers.

Current Proposal

For each of these cases, the desired approach is to capture an element's snapshot to avoid a visual pop to the extent possible. For example, clip rects and opacity from the ancestor chain can be accumulated and flattened into the element's snapshot but clip-paths may be harder to implement.

The behaviour to describe how content from the two documents is drawn together during the transition is as follows:

- We assume that the dimensions for the visual viewport for both documents is the same. If this changes, for example because the user resizes the browser window before the transition finishes, the transition is aborted and we switch directly to the final state.

- During the transition, the UA creates a new root element (hidden from script), referred to as *TransitionRoot* which establishes the uber root stacking context and containing block. By default, all shared elements are direct descendants of the *TransitionRoot* (including root elements from both documents).

- We compute the position/transform (alongside the snapshot) of shared elements in the old document when a navigation is initiated. And for shared elements in the new document on first paint. These map directly to the position/transform in *TransitionRoot* since the viewport size hasn't changed. Animating the elements is covered under Specifying Animations.

- It's unclear what the paint order of these elements should be.

Specifying Animations

Finally the API needs to provide the syntax for defining the animation. We'd like to layer this on top of existing Animation APIs (CSS transitions or Web Animations). A few considerations for this are:

- We need a way to establish pairing between elements for paired transitions.

- The API needs to explain the concept of transitioning between 2 elements with different sizes and pixel content. The operation is somewhat similar to a css image cross-fade. The default is to interpolate the size of the images as the transition progresses but there are use-cases where clipping makes more sense, or no size transition at all. So this needs to be eventually customizable.

- We'd prefer to start with higher level patterns (like slide) for single element transitions. But eventually allow creating such patterns by specifying a combination of interpolating properties (like transform, opacity). In either case, knobs needed are similar to most animations: delay (with respect to the universal timeline), duration, easing curve.

Current Proposal

The current proposal uses a new dictionary for the above. The animation controls work at a granularity of one single element or paired element transition. For example, for a paired transition the developer can specify a duration for the complete effect : interpolating the size of the two images, a cross-fade between the pixels and animating the element's position. Along with a delay for when this animation starts with respect to other elements.

But they can't control the duration/delay for each of the individual property interpolations. We do want to expose this control going forward and would like to base this on existing web primitives. For example, using the keyframes syntax in WebAnimations (see feature request).