Throttling Blink's rendering pipeline for hidden content

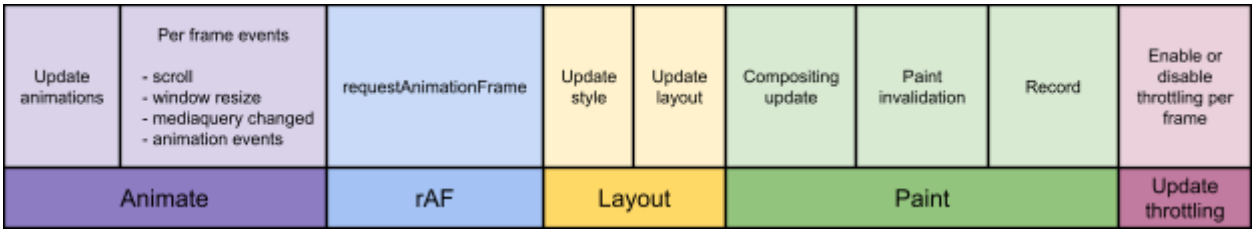
skyostil@

August 28th, 2015
Tracking bug: 487937
Patch: https://codereview.chromium.org/1364063007/

Introduction

Pages with many animated elements – often advertisements – can end up having a poor user experience or low power efficiency because animation and other work is performed for the entire page instead of just the visible parts. This document proposes an improvement where we limit the amount of computation we do for frames which are outside the visible viewport bounds.

Algorithm



The diagram above (adapted from Resize Observers) shows the main stages of Blink's rendering pipeline plus a new stage for configuring per-frame throttling. Each of the stages processes the whole frame tree recursively. The proposal is to terminate the recursion at each stage if:

1. the frame to be processed is outside the visible viewport bounds, and
2. the frame to be processed is in a different security origin compared to the current frame.

In practice, this will stop all rendering pipeline related processing in each respective stage for cross-origin frames which are outside the viewport.

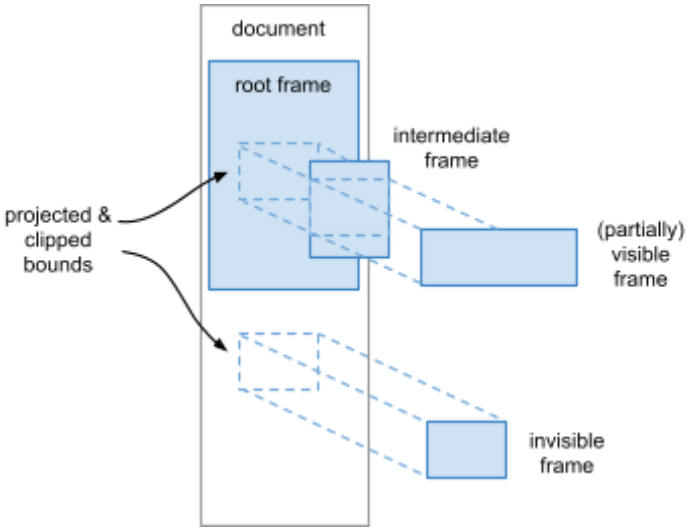
Cross-origin requirement

The visibility determination is only done for cross-origin frames to reduce the chances of breaking content which could synchronously observe the suspension of rAF callbacks and other side effects the pipeline in the affected frame. Cross-origin frames can only

communicate asynchronously, so existing content must already deal with deferred responses.

Visibility determination

Every time a frame's geometry changes, we recursively project its bounds all the way up to the root frame view. At each level the bounds are clipped to the intermediate frame's viewport. If the bounds become empty at any point, this means the frame isn't visible in the root viewport.



Note that we use exact visibility determination instead of relying on the compositor prepainting region. This is because the compositor currently uses an aggressively large (several kpixels) prepainting region, limiting the effectiveness of this optimization. As a future improvement a "Soon" visibility state could be added to reduce the chances of seeing stale content while scrolling.

Throttling update stage

The new throttling update stage ensures that throttling is enabled or disabled at controlled times. This is because the visibility of a frame could change at any point during the animate, rAF or layout stages, but it is not possible to start the pipeline for a frame in the middle without also going through all the preceding stages. Instead, the new visibility information is applied to throttling at the very end of the pipeline; if any frame became unthrottled because of this, we schedule another pipeline update to guarantee eventual consistency.

Forced layouts

Sometimes Blink needs to perform a forced layout (a.k.a. lifecycle update) to obtain consistent geometry information. This happens for example during hit tests. In these cases we need to run the style, layout and compositing update parts of the pipeline for every frame including throttled ones.

Future improvements

- Instead of exact visibility determination, use the compositor's prepainting region via some kind of `PaintObserver`. The compositor currently uses an aggressively large prepainting region, limiting the effectiveness of this optimization. If that can be reduced, we could add an intermediate "`Soon`" visibility state.
- A latency improvement (suggested by chrishtr@) would be to perform a layout pass after the animation update to see if any new frames became visible, and, if so, update their animations. This would eliminate one frame of rAF latency when a layout change makes a previously hidden frame visible. Note that because the compositor still runs asynchronously, this wouldn't completely eliminate the possibility of seeing old content. For simplicity this is left out from v1.

Open issues

- Do cross domain iframes with zero size, display: none or similar need to be considered visible? smfr seems to think so.
    - The current implementation does not throttle "display: none" iframes, but always throttles 0x0 iframes.
- Can we somehow safely extend this to same origin frames?

Resources

- requestAnimationFrame prollyfill with visibility support: https://github.com/greggman/requestanimationframe-fix.js
- HTML processing model: https://html.spec.whatwg.org/multipage/webappapis.html#processing-model-9

Research

- How did Firefox implement this?
  - Tracking bug, patch
  - Shipping in Firefox 40
  - Throttling done a per document level
  - Throttles to 1 fps instead of completely disabling rAF
  - Visibility check is based on whether the document was painted in the last frame. This can cause one extra rAF to go through after the iframe has left the screen. Also, one frame is dropped when the iframe becomes visible again.
  - Problems with first paint – code thinks the document is still invisible
  - How does this work when we don't paint at all? Not sure, but it seems to work. Maybe they always trigger paint in rAF?
- How far along are the various visibility APIs and can we reuse them?
  - Well, the page visibility API is already shipping.
  - The API is limited to to tracking the visibility of the root Document, but there's a w3 thread by seth@mozilla.com (who also implemented rAF throttling) about extending it to iframes.
    - Earlier identical proposal on public-web-perf. Issue: some APIs are tied to page visibility in their specs.
    - Alternate suggestion: <iframe autopause>
    - Discussion seems to have stalled
  - Element visibility API doesn't seem to exist
- How is visibility determined for animated GIFs?
  - https://codereview.chromium.org/1026823002 added the notion of a delayed paint invalidation for images
  - https://codereview.chromium.org/1145643002 extended it to handle animated background images
  - LayoutBox/Image::imageChanged() sets the invalidation reason to PaintInvalidationDelayedFull
  - LayoutBox::invalidatePaintIfNeeded() checks LayoutBox::intersectsVisibleViewport() and keeps the invalidation reason as DelayedFull unless the box intersects the viewport
  - The GIF animation timer only ticks once and is restarted once the image gets drawn again
  - Uses the skewport/extended viewport by virtue of the compositor drawing a little outside the screen
- How does the resource scheduler determine visibility?
  - Works on a per-tab basis (client == tab)
  - Tab is considered to be active if it is visible or audible
  - Doesn't use per-element visibility or spatial prioritization -- except for images.
- How expensive is it to compute visibility?

    - Doesn't seem to be too bad since we're doing it for every animated image every time we paint.
    - All FrameViews are also already notified when their geometry changes.
- How do we avoid checkerboarding?
    - Firefox didn't seem to worry too much about this since their rAF can be delayed by one frame after the iframe becomes visible.
    - Made a small test page that measures the time between rAFs. Seems to drop up to two frames on Firefox when becoming visible.
- Do we want to suspend rAF completely or just throttle it?
    - Intuitively doing animation work for hidden content doesn't feel sensible. Limiting this to cross origin frames should be safe.
- Architecture: how do we reconcile this with spatial scheduling (timers, loading tasks)?
    - FrameView viewport visibility observers seem like a good building block.