Overriding viewport bounds and scale for screenshots

PUBLIC

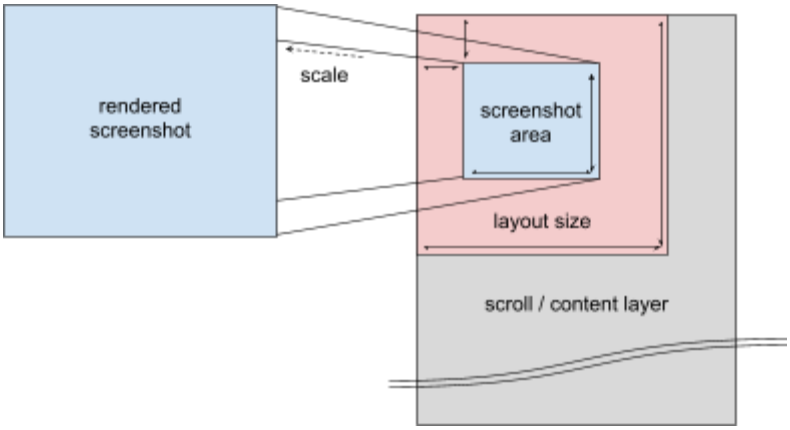
Authors: eseckler@

Motivation

For Headless Chrome, we would like to provide users with a flexible way to take scaled screenshots of specific regions of a page. We plan to support our use case by resizing and positioning the renderer's layout viewport and resizing, positioning, and scaling its visual viewport. Screenshots can then be taken from the rendered image as seen in the visual viewport.

By utilizing the viewports for this purpose, we also effectively control the compositing layers. Thus, we ensure that only the part of the page relevant for the screenshot is rendered.

Example: We want to layout a page with layout size 1200x800px and take a screenshot of an advertisement at (x=400, y=2500, width=200, height=100) at 2x scale. For this, we override the layout viewport size to 1200x800px, the layout scroll position to (0, 2500), visual viewport scroll position to (x=400, y=0), scale to 2.0, and size to 200x100px.



Approach

Current commands in the DevTools Protocol allow emulation of frame size and certain other device metrics (`Emulation.setDeviceMetricsOverride`), as well as taking screenshots of the visible window contents (`Page.captureScreenshot`). We will extend the existing override command with parameters for the visual viewport size. Further, to support positioning the layout and visual viewport, we are adding an additional override command that can be used to override the scroll position of the layout viewport and the scroll position and scale of the visual viewport. These overrides should be enforced in ways that prevent both programmatic and user interactions with the page from altering the overridden parameters.

Enforcing scroll overrides

We enforce scroll overrides within Blink in `FrameView` and `VisualViewport` by overriding the values returned by the respective `minimumScrollPosition/Double()` and `maximumScrollPosition/Double()`. When new scroll positions are applied within Blink, they are consequently clamped to the overridden values.

Enforcing scale overrides

We use a similar mechanism to constrain the minimum and maximum page scale to the overridden value: We set the min/max scale of the user-agent `PageScaleConstraints` in the `FrameHost's PageScaleConstraintSet`. These constraints are enforced when new page scales are applied within Blink.

Implementation

There's a patch demonstrating this approach. It adds `ScrollAndScaleEmulator`, which contains overriding logic for viewport scroll positions and page scale and is queried from FrameView and `VisualViewport`.

Risks

1. Screenshots are taken from content visible in window.

Resizing the visual viewport in Blink alone will not be enough to ensure that the screenshot taken will correspond to the viewport size, since screenshots are taken from the browser's compositor layer corresponding to the `RenderWidgetHost`'s window. Thus, we plan to resize this (Aura) window to match the visual viewport's dimensions, e.g. by resizing or applying an override in `RenderWidgetHostView`. This could be controlled e.g. through an additional boolean `resize_window` parameter for the `setDeviceMetricsOverride` command. Taking screenshots directly from the renderer rather than the browser is not realistic, since out-of-process i-frames and GL content are not visible to the renderer.

2. Visual viewport will eventually be exposed to page.

The work-in-progress VisualViewport API will expose the visual viewport position/scale to the page. Thus, the page can change its appearance when we use the visual viewport for screenshotting. There are two situations here:

a) User wants to take a screenshot of a genuine visual viewport setup. In this case, the page should be allowed to react to the change in viewport position/scale.
b) User wants to take a screenshot of an area as previously shown on the frame (e.g. an advertisement). In this case, the visual viewport is only used as a means to select this area for the screenshot and the page should not be allowed to react on the change.

Since we envision b) to be likely, we plan to add a way to optionally hide the change of visual viewport position/scale (that is due to overrides) from the page.

3. Additional controls for auto-expanding layout viewport.

Another likely use case is to take a screenshot of the “whole” page. For this purpose, we plan to add a mechanism to auto-expand the layout viewport to the size of the page. This will happen separately for its width and height in two steps, to a) allow the page to re-layout in between and b) allow auto-expansion with one fixed dimension. We intend to realize this mechanism using multiple subsequent invocations of the `setDeviceMetricsOverride` command.

Corner cases to support

The following corner cases should be supported and tested:

- Visual viewport that is larger than the frame (to support screenshots of content larger than the layout frame).
- Visual viewport that is larger than the page contents (to support screenshots of a fixed size).
    - Ensure that the background color for the area outside the page contents is set correctly.
- Auto-expansion of layout frame to page contents (width, height, or both).
- Auto-clipping of visual viewport to page contents (width, height, or both).

Override implementation issues (mostly solved)

1. User scroll events may be handled by the compositor instead of Blink.

User inputs for scrolling can be handled by the compositor thread, instead of the main renderer one. This means that the compositor can schedule scrolling animations without asking Blink to clamp the target scroll positions to overrides first. This can result in two behaviors depending on our implementation:

a) Blink-side discards the eventual commit of a new scroll position from the compositor, because the new position (after clamping) is the same as the old one it knows about. Consequently, the compositor scroll is not prevented.
b) Blink-side re-applies the clamped new scroll position on compositor commits. Compositor scrolls may be animated before this can happen, so the content may jump "back" after the re-clamping.

Solution

We are solving this problem by disabling threaded scrolling (through `Settings`) while a scroll position override is active. In the process, we also discovered and fixed a situation in which scrolling could still happen in the compositor, even if threaded scrolling was disabled.

However, this solution does not yet prevent changes to the scroll position by the compositor in the context of pinch-zooming (crbug.com/626277).

2. The effective minimum page scale influences layout viewport size.

`WebViewImpl` scales the main frame (layout viewport) size to the inverse of the effective minimum page scale (e.g. minimum scale 2.0 => half frame size). As a result, fixed-pos elements are placed differently, and scrollbars appear in the wrong places.

Page with scale 1.0 (original), scale 2.0 (expected), scale 2.0 (actual).

Solution

We modify the behavior of `WebViewImpl::mainFrameSize()` if a scale override is active. In this case, we reconstruct the original `PageScaleConstraintsSet` without override to calculate the size.

Alternatives considered

1. Set position and scale directly before screenshot, without applying an override.

Overriding the positions and scale of the two viewports ensures that user interactions and scripts running on the page cannot modify these positions. It also allows us to enforce a scale outside of the `PageScaleConstraints` set by page and/or browser. As a consequence, this may result in viewport setups that wouldn't normally be possible. However, it gives us the flexibility to repurpose the visual viewport to select and scale an arbitrary area of the page (e.g. an advertisement). Setting the position/scale without applying overrides would not allow us to do this.

2. Take partial screenshot rather than adjusting viewports.

As an alternative to viewport repositioning, we could simply set the visual viewport size to the size of the visible page contents and take screenshots of a specified sub-area shown in the viewport. The advantage of this approach is that it circumvents risk #2 described above. However, particularly for large pages, rendering the whole page (possibly at large scale), will incur a significant overhead. By positioning the visual viewport instead, we also control the compositor layers. Thus, clipping will occur and only the relevant area will be rendered. We believe that the additional effort required to address risk #2 is worth the savings gained in this way.