Accelerated overflow scrolling

(without stacking containers)

hartmanng@, vollick@

tl;dr

With this plan, we can get overflow:scroll div's impl-side scrolling in ***all*** cases.

Problem:

Accelerated overflow scrolling is currently accomplished by rearranging the RenderLayer and GraphicsLayer trees. When accelerated, the RenderLayer that scrolls overflow gets two additional GraphicsLayers layers created by its backing:

1. The scrolling layer, which is simply a clip for
2. The scrolling contents layer, whose position is updated via the impl-side scrolling machinery.

However, it is often the case that the scrolling RenderLayer is not a stacking context, and as a result, it often happens that its descendants are siblings in the same stacking context. This means that the descendants will not have their associated GraphicsLayers parented under the scrolling contents layer and they will not scroll together. It's also commonly the case that the RenderLayer isn't the containing block for some of its descendants. It could, for example, have a fixed positioned descendant whose containing block is the viewport and will therefore not respect the scrolling RenderLayer's clip.

The current solution to these problem is to 'promote' the scrolling RenderLayer to stacking context and containing block status. Since it's not necessarily a stacking context in the css sense, we've introduced the notion of a 'stacking container' to describe a RenderLayer that owns z-order lists. If the scrolling layer is made a stacking container, the parenting is set up in such a way that all layers scroll together. But this has a number of unfortunate side effects:

1. Any out of flow descendants that used to extend beyond the clip established by the scrolling layer will now be clipped (because any descendant of a clipping GraphicsLayer, *must* be clipped by it, currently).
2. Any non-descendant that used to positioned between descendant layers will now 'pop out' and appear either above or below the scrolling layer and all its descendants.

We have a large amount of complex code to detect these situations and prevent opting into the accelerated path.

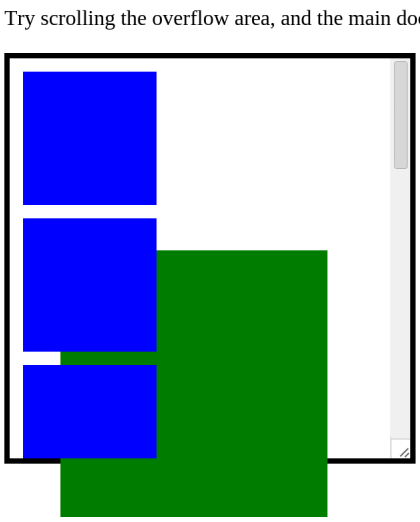
See also: slide deck from a brief talk about what's wrong with the current approach, and why the new approach solves the major issues.

Solution:

Do away with stacking containers and instead rely on the Chromium compositor to scroll the

required layers on the impl thread and clip them appropriately. To do this we'll introduce the notions of a 'scroll parent' and a 'clip parent'. A layer scrolls with, and is clipped by its scrolling parent. A clip child is clipped by its clip parent and its clip parent's ancestors, but not by any intervening layer between the child and parent. Roughly speaking, both the scroll parent and the clip parent correspond to the layer's containing block.

Take for example this page: http://smfr.org/misc/css/stacking/css-escher.html



In this case, the blue boxes scroll when the scroll thumb is dragged. Their scroll parent is the overflow scrolling div (its scrolling content layer, specifically) and they inherit its clip. The green box, on the other hand, is fixed positioned and does not scroll when the thumb is dragged. Its scrolling parent is the viewport, and so it does not appear to move nor does it appear clipped.

Here are the working aggregate patches:

- blink: https://codereview.chromium.org/20991002/
- cc: https://codereview.chromium.org/20990002/

Blink Implementation Details

We have three main tasks to accomplish on the blink side of the fence:

1. We have to ensure that the clip children and scroll children get promoted (otherwise the cc won't have anything to work with).
2. We have to set up the scroll and clip child/parent relationships.
3. We have to ensure that our scroll children are clipped properly when they scroll on the impl-thread.

Promoting scroll and clip children

Before describing how we identify scroll and clip children, we should establish some terminology. By *DOM tree* I mean the layer tree formed by the parent/child relationships of the RenderLayers (it *very* roughly corresponds to the DOM topology) and by *stacking tree* I mean

the layer tree formed by traversing the RenderLayers' z-order and normal flow layer lists.

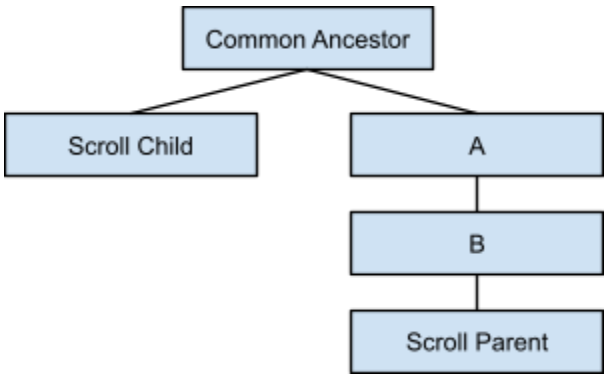
Using that terminology, a scroll child is a layer with a scrolling ancestor in the DOM tree that is not an ancestor in the stacking tree. A clip child is any layer with a containing block *above* an ancestor clipping layer in the stacking tree (in the code, we refer to these nodes as *unclipped* descendants).

Setting up scroll and clip relationships

The scrolling coordinator calls WebLayer::setScrollParent and ::setClipParent when needed.

Ensuring that scroll children are properly clipped

We treat our scroll parents as clip parents. But it's not as simple as it sounds. The compositor currently traverses the layer tree in paint order when updating clipping, and there's no guarantee that our scroll parent will precede us in paint order so we can't use its clip information. Here's an example of a case where we'd get in trouble with the current code.



Why can't we just process A before Scroll Child? Because we produce a number of layer lists during our traversal of the tree, and if we traverse out of order, we'll get them jumbled. Fortunately, these lists are easy to sort, so this turns out not to be an issue in practice.

Implementation requirements:

1. Introduce the notion of a 'scroll parent' to WebLayer and cc::Layer. The compositor will use this relationship when it applies scroll offsets and clips.
2. Ensure that every descendant RenderLayer of a non-stacking context RenderLayer is composited and has its scroll parent set appropriately.
3. Ensure that out-of-flow positioned descendants are promoted if they need to be counter scrolled.
4. Update the cc to make use of the scroll parent.
5. Implement layer-squashing, and UMA stats to measure its efficacy. This plan can easily result in a layer explosion.

Testing requirements:

1. A non-descendant scrolling between two descendants.
2. A descendant that extends beyond the scroll layer's clip.

3. http://smfr.org/misc/css/stacking/css-escher.html
4. All of the above with non-RenderLayer content (e.g., text) in the scrolling RenderLayer.
5. A cc perf test demonstrating that we don't make cc way slower. (Note, given that this approach has little impact on cc/ performance, our real concern is with the efficacy of layer squashing).

Patches

cc patch series

(here's the aggregate, if you want to see it all together)

1. [optional/p4] Add trace event instrumentation for comp-scroll and impl-scroll
    a. As I was putting these patches together, I added a number of trace events that I thought might be generally useful. I've put them together just in case, but it's not important they land.
2. Plumbing for clip and scroll parents
    a. Just ensures that the scroll and clip parent/child relationships make their way to the impl thread correctly.
3. Make use of clip and scroll parent relationships
    a. Uses the scroll and clip parents to update clips and scroll deltas.
4. Fix painting for non-main-frame scrolling layers when not impl-painting
    a. When we're not impl-painting, scroll children suffer from the same prepainting/prioritization woes that small animated layers do. Treat them the same in this case.
5. Ignore should-scroll-on-main thread reasons when not applicable
    a. Currently, an impl-scroll can be canceled for bogus reasons. We bubble our way to the top of the impl tree (even after we've found a scrolling layer) to see if there's any layer forcing us to scroll on the main thread. One way we can be forced to scroll on the main thread is if a layer should_scroll_on_main_thread(). However, these reasons only apply to main frame scrolling. If the main frame cannot scroll (as is the case in gmail), then these reasons are not applicable and should be ignored.

blink patch series

(here's the aggregate, if you want to see it all together)

1. Add layout tests for universal overflow scrolling.
2. Promote extra layers to remove corner cases from overflow:scroll fast path
    a. On the Blink side, we need to promote all layers which are unclipped, and also all layers which are the direct children of a non-stacking context overflow-scroll div.
3. Promote all siblings after an unclipped descendant in stacking order
    a. We don't overlap test during impl scrolling. To ensure we don't get incorrect behavior, we must promote layers that could potentially promote due to overlap during a scroll. To fix this, we also have to go ahead and promote any layer that is a sibling of an unclipped descendant, and also comes after the unclipped descendant in stacking order, just in case they may overlap during the scroll.
4. Set Chromium-side defaults for compositorDrivenAcceleratedScrollingEnabled setting
    a. The compositorDrivenAcceleratedScrollingEnabled setting is a blink-only temporary setting to keep these patches disabled by default until it's all working. Unfortunately, during layout tests, the value doesn't get reset unless we add a bit of Chromium-side plumbing.

5. Set up clip and scroll parents on the blink side
6. Make composited scrolling codepaths co-operate
    a. Since we don't have a functional layer-squashing solution yet, it makes sense to leave in the logic for the stacking container concept for the short term, and use it whenever possible. If a layer cannot become a stacking container, then we fall back to using the new codepath that's being set up in all the above patches.
7. Reset Chromium-side defaults for compositorDrivenAcceleratedScrollingEnabled setting
    a. Due to the previous patch, the Chromium-side default for this value is now wrong (since we now want to turn this *on* by default), so we need one last one-liner to enable the setting everywhere.