Paint invalidation in Slimming Paint (V1 and V2)

chrishtr@chromium.org
wangxianzhu@chromium.org

Paint invalidation in SPv2

The paint invalidation tree walk should be removed for slimming paint, v2 for these reasons:

1. Complexity: it introduces complexity into the LayoutObject class hierarchy for a non-layout task.
2. Bugs: it is known to have bugs due to the mismatch between the order of recursion in the LayoutObject tree walk and the performance optimizations in PaintInvalidationState, which led to hacks to remove those optimizations in many cases, in particular for positioned elements.
3. Performance: due to the hacks mentioned in #2, paint invalidation is slower than it needs to be. Furthermore, since it is an extra tree walk, it will take as long as that tree walk takes to complete.

The input to paint invalidation is a set of bits on LayoutObjects indicating whether they may or definitely need paint invalidation. The output is:

1. A marking of which DisplayItemClients are actually invalidated, and
2. Recordings of visual rect invalidations for each DisplayItemClient in the backing of its paint invalidation container.

Currently, paint invalidation computes the visual rect of each LayoutObject which may need paint invalidation, then compares this against the size and position of its visual rect from the previous frame, in the space of its paint invalidation container. If they differ, then it records an invalidation of the visual rect (#2) and invalidates the DisplayItemClients which belong to that LayoutObject.

In slimming paint v2, there will be no concept of a paint invalidation container. Instead, DisplayItemClients will be invalidated, and that will automatically lead to the correct invalidations for paint and the compositor. In place of a visual rect in the space of a paint invalidation container, we will invalidate a LayoutObject if

- It's marked shouldDoFullPaintInvalidation
- or marked mayNeedPaintInvalidation and any of these parameters differs from the prior frame:
    - Transform property tree state (node and path to the root)
    - Paint offset relative to the containing transform node
    - Overflow, border and or/content box size (in general all three, but often fewer)

- or forced checking of paint invalidation by some ancestor, and any of the above parameters changed
    - This happens in spv1 when location of an object changes. Do we need to keep this for spv2?
    - ~~[Done] With this we can handle setShouldDoFullPaintInvalidationIncludingNonCompositedDescendants, etc. without tree walk during marking, but mark the root of the object needing subtree paint invalidation.~~

Object invalidation -- Code ready, needs testing

This will be done in PrePaintTreeWalktree walk:

- Walk the layout tree and invalidate DisplayItemClients using the method described in the previous sections. Call setDisplayItemsUncached() on DisplayItemClients that needs paint invalidation.
- Update needsPaintPhase and needsRepaint flags on PaintLayers.

Raster invalidation

This will be done during painting by PaintController based on the object invalidation information and visual rects of display items.

For non-paint-property changes -- Initial version landed

For each new paint chunk, find the matching old chunk. If not found, issue a raster invalidation rect covering the whole new chunk. If found:

- For each DisplayItemClient in the previous paint chunk but not in the new paint chunk, invalidate the previous visual rect of the client.
- For each DisplayItemClient in the new paint chunk but not in the previous paint chunk, invalidate the current visual rect of the client.
- For each DisplayItemClient marked needing paint invalidation, invalidate the visual rect (for full paint invalidation) or the difference between the previous visual rect and the current visual rect (for incremental paint invalidation).
- <TBD> Potential optimization for display item reordering in a chunk. Currently we have paint invalidation flags set on the objects, so the above algorithm should work.

For paint property changes

We will handle these changes during compositing.

Identification of paint chunks -- Initial version landed

The method described above requires each paint chunk to have a unique id so that we can match old and new paint chunks.

We can use DisplayItem::Id as the ids of paint chunks. The client is the object creating the paint chunk. The type can be paint phase based, or any applicable special type e.g. when issuing a paint chunk for clipping we can use the clipping types.

To detect deletion of a paint chunk client and avoid treating a new client created at the same address of an old client, we can give a new DisplayItemClient a special cache generation value. A paint chunk with a client with the special cache generation will be treated as a new paint chunk which doesn't match any old chunk.

Paint chunks created during paint controller skipping cache may not have unique ids. They are marked to be treated as new chunks not matching any old chunk, and will invalidate the whole bounding box.

However, for paint chunks created in a validly cached subsequence, we will not check their changes and issue any rect paint invalidations even if some of them skipped cache.

Notes

All of the above rects are in the space of containing transform node (paintOffset + localOverflowRectForPaintInvalidation).

Need to map the rects (using GeometryMapper) into space of the parent transform node if a transform node is removed during layerization.

Clippings are not applied on visual rects of display items. We expect that later stages will handle the clippings correctly.

Can we use paint invalidation flags to trim paint property tree walking? -- Done by pdr@

This need partial update of the property trees: we should be able to update a node without recreating the subtrees of it.

Paint invalidation optimization for offset-only changes

http://crbug.com/526189
This can be achieved if the offset is using transform, then we will treat this as a paint property change. For paint offset change under a transform node we can just treat it as a full paint invalidation of the display item client, but defer the decision of raster invalidation to PaintArtifactCompositor after layerization.

Incremental paint invalidation

We are doing incremental paint invalidation for an object if

- the object is not marked needing full paint invalidation
- its location/paint-offset doesn't change
- its paint invalidation size changes

- the newly exposed/unexposed parts of paint invalidation rect covers all parts needing re-rasterization

This helps to reduce the amount of re-rasterization when an object is resized, if possible.
We can keep this given that we still issue rect-based paint invalidation in blink.

Rect-based paint invalidation

This corresponds to ObjectPaintInvalidator::InvalidatePaintRectangle().

- Convert to full paint invalidations? Need to evaluate for each situation.
- Keep them: we need to save the rects for the DisplayItemClient, and issue raster invalidations during PrePaint

Scenarios

Non-paint-property changes

	Own display items	Subtree	Raster invalidation rectangles (or perhaps regions)
Full invalidation	Invalid	Unaffected	The previous and current visual rects
Resize not needing full invalidation (unclipped)	Invalid	Unaffected	Difference between previous and current visual rects
Paint offset change (without a transform node)	Invalid	Forced to check invalidation of descendants that may be affected by the paint offset change	Raster invalidation is not always necessary. In case that paint offset changed in the current chunk but the chunk is finally squashed into parent chunk, if the paint offset doesn't change we don't need invalidation. So defer the invalidation to layerization stage.
Reordering within a chunk	For now we fully invalidate reordered items. We can avoid this.	For now we fully invalidate the recordered subtrees. We can avoid this.	For now same as full invalidation. We can detect recordering and invalidate if needed in PaintController.
Moving from one chunk to another chunk	Valid	Unaffected	The previous visual rect in the original chunk and current visual rect in the current chunk.

(in case not invalidated)			

Paint property changes on the original tr~~ee node~~

Handle this during compositing (PaintArtifactCompositor).

	Own display items	Subtree	Invalidation rectangles (or perhaps regions)
Effect	Unaffected	Unaffected	Bounding box of the display items in the effect subtree
Transform	Unaffected	Unaffected	Bounding box of the display items in the transform subtree
Clip	Unaffected	Unaffected	The previous and current of clip rectangles (or perhaps regions)

- Record conditional invalidation rects, and invalidate them if needed in PaintArtifactCompositor
- Use union rects in cc:DIC begin transform

Paint property tree structure changes

This affects on which paint chunk we should invalidate the previous visual rect of the affected display items. The logic is similar to that for SPv1 handling paint invalidation container, stacking context and composited layer tree changes.

Sometimes moving between paint chunks doesn't necessarily need paint invalidation because the paint chunks may be merged into a common parent paint chunk.

New paint invalidation in SPv1 (Done)

As an intermediate step, before SPv2 is ready, we can use PaintPropertyTreeBuilder in SPv2 to fix the current issues of paint invalidation, and prove some of the concepts presented in this document earlier.

Some design points:

- Track paint invalidation container for normal objects, absolute position objects and fixed position objects with PaintPropertyTreeBuilder;
- During PaintPropertyTreeBuilder tree walk, call LayoutObject::invalidatePaintIfNeeded() for each object, with PaintInvalidaitonState constructed based on the current PaintPropertyTreeBuilderContext (for paint invalidation container, paint offset, etc.);

    - paint invalidation container and paint offset will be selected from those tracked separately for normal objects, absolute position objects and fixed position objects according to the position style of the current object, so we can always get the correct data for the current object, avoiding the mismatching paint invalidation container issue of the original paint invalidation code.
- Don’t change how LayoutObject::invalidatePaintIfNeeded() works;
- Handling of special cases that were handled with ForceHorriblySlowRectMapping:
    - Keep them in the first CL
        - but the ones created in invalidateTreeIfNeeded and invalidatePaintOfSubtreesIfNeeded are automatically skipped because we no longer call them. This includes cases such as mismatched paint invalidation container, and paint offset issue of positioned object inside of positioned inlines.
    - Try to avoid ForceHorriblySlowRectMapping for transforms;
    - Try to avoid ForceHorriblySlowRectMapping for other cases. We may also have to keep some of them.