CompositeAfterPaint

Xianzhu Wang (wangxianzhu@google), Philip Rogers (pdr@google), Chris Harrelson (chrishtr@google), Stefan Zager (szager@google), Daniel Libby (dlibby@microsoft)

The blink main thread produces a list of display items which are split up into *composited* cc::Layers which can be independently moved off the main thread, such as for scrolling or composited transform animations. CompositeAfterPaint is a project to decide what to composite after running the html/css paint algorithm, rather than before.

Launch bug: https://crbug.com/471333
Status: launched via finch in M94+ (Oct 28, 2021) and enabled by default in 97.0.4684.0.

Goals

- Correctness
  - Fixes the fundamental compositing bug (crbug.com/370604). This architectural bug results in content drawing out of order and is due to the requirement of a PaintLayer, which affects paint order, in order to have composited content. Fixing this will improve compatibility with other browsers.
  - Stop querying compositing state in the wrong lifecycle phase (crbug.com/1007989). This class of bugs (e.g., crbug.com/625676) is due dependencies of the style and layout system on compositing, which may be stale.
- Performance
  - Improve the performance of deciding compositing through an improved algorithm. Compositing is frequently the source of long frames (e.g., https://crbug.com/969683#c5) and we have data that compositing is the largest component of long (>5ms) frames.
  - Reduce GPU memory by making better compositing decisions. This will improve jank, latency, and stability metrics (see: GPU memory ablation study).
  - Enable compositing of arbitrary content such as SVG (crbug.com/666244: 137 stars), blinking cursors, and gifs.
  - Enables moving compositing decisions off the main thread (design doc).

Implementation

The existing paint lifecycle step

Paint produces a list of DisplayItems in paint order (back-to-front). There are two primary types of DisplayItem: DrawingDisplayItem which has a PaintRecord, and ForeignLayerDisplayItem which has a reference to content drawn outside of blink such as a hardware accelerated video.

DisplayItems are grouped into PaintChunks which are contiguous DisplayItems that share the same property tree state (transform, clip, and effect paint property nodes). PaintChunks are the smallest atom of compositing. Pre-CompositeAfterPaint, these PaintChunks are stored on the pre-computed composited layers (GraphicsLayers). With CompositeAfterPaint, a single list of PaintChunks is produced for the page.

Deciding compositing

CompositeAfterPaint introduces an algorithm for taking all PaintChunks on the page and grouping them into cc::Layers (PaintArtifactCompositor::LayerizeGroup). This has to make tradeoffs between GPU memory and reducing the costs when things change. The new algorithm aggressively merges PaintChunks, in-order, into the smallest possible set of cc::Layers, with each PaintChunk belonging to some cc::Layer. New cc::Layers are required when PaintChunks cannot be merged, such as when there are direct compositing reasons on property nodes, or when composited content could interleave (also known as "overlap testing"). Once the set of cc::Layers is determined, all PaintChunks for each cc::Layer are combined into a final display item list (see: PaintChunksToCcLayer).

Results (experiment data)

The following data is from a 10% experiment on M94 stable on Android, Windows, and Mac, and a 1% experiment on ChromeOS. Raw data and UMA links are available here. Blue backgrounds indicate statistically significant results ($p < 0.05$).

GPU memory

	Android	Win	Mac	CrOS
Memory.GPU.PeakMemoryUsage2.PageLoad @ 50th	-2.82%	-1.21%	-1.73%	-0.41%
Memory.GPU.PeakMemoryUsage2.PageLoad @ 95th	-3.61%	-2.10%	-1.97%	0.23%

Note: GPU memory is not included in Memory.Gpu.PrivateMemoryFootprint or Memory.Total.PrivateMemoryFootprint. On unified memory architectures such as Android, GPU memory is using the same chips as system memory.
Reasons for this improvement: algorithm does not need to composite scrollers just due to composited descendants, improved merge algorithm considers memory, better cull rect that avoids painting clipped-out content.

Speed

	Android	Win	Mac	CrOS
Event.Latency.ScrollUpdate.Touch.TimeToScrollUpdateSwapBegin4 @ 75th	-1.67%			
Event.Latency.ScrollUpdate.Touch.TimeToScrollUpdateSwapBegin4 @ 95th	-1.22%			
Event.Latency.ScrollUpdate.Touch.TimeToScrollUpdateSwapBegin4 @ 99th	-3.54%			
Event.Latency.ScrollUpdate.Wheel.TimeToScrollUpdateSwapBegin4 @ 75th		0.14%	-1.02%	-1.02%
Event.Latency.ScrollUpdate.Wheel.TimeToScrollUpdateSwapBegin4 @ 95th		-1.10%	-3.68%	-3.43%
Event.Latency.ScrollUpdate.Wheel.TimeToScrollUpdateSwapBegin4 @ 99th		-5.64%	-9.26%	-7.86%

The Android metric, Event.Latency.ScrollUpdate.Touch.TimeToScrollUpdateSwapBegin4, is a guardian metric and a Blink 2021 objective is to improve it by 10% at 75th and 95th.
Reasons for this improvement: algorithm is faster, leading to improved overall main frame time. GPU memory reduction is known to correlate with improvements (gpu ablation study).

	Android	Win	Mac	CrOS
PageLoad.InteractiveTiming.InputDelay3 @ 50th	-2.07%	-0.70%	-0.03%	-1.21%
PageLoad.InteractiveTiming.InputDelay3 @ 75th	-4.07%	-2.87%	-0.19%	-1.93%
PageLoad.InteractiveTiming.InputDelay3 @ 95th	-2.70%	-3.01%	-2.20%	-2.72%

PageLoad.InteractiveTiming.InputDelay3 is a guardian metric, and a Blink 2021 objective is to improve by 10% at 75th and 95th.
Reasons for this improvement: algorithm is faster, leading to improved overall main frame time.

	Android	Win	Mac	CrOS
WebVitals.FirstInputDelay @ 50th	-1.65%	-0.41%	-0.26%	-0.51%
WebVitals.FirstInputDelay @ 75th	-2.92%	-0.65%	-0.30%	-1.66%
WebVitals.FirstInputDelay @ 95th	-0.40%	-1.25%	-1.19%	-1.24%

Reasons for this improvement: algorithm is faster, leading to improved overall main frame time.

PageLoad.Clients.Ads.AdPaintTiming.NavigationToFirstContentfulPaint3 regresses significantly. This is actually a progression due to more accurately calculating the metric (see: https://crbug.com/1236136) and we have approval (link) to proceed with this metric change. Non-ad NavigationToFirstContentfulPaint3 is unchanged.

Stability

	Android	Win	Mac	CrOS
User.ClientCrashProportion.RendererCrash	-1.79%	-0.65%	-0.11%	-1.10%

Battery

	Android	Win	Mac	CrOS
Power.CpuTimeSecondsPerProcessType, total	-1.32%			
Power.CpuTimeSecondsPerProcessType, renderer	-2.66%			
Renderer.TotalCPUUse2 (sum)		-2.04%	-1.99%	-1.64%
CPU years saved (years / day)	526	650	50	25
Power.ForegroundBatteryDrain.30SecondsAvg (trimmed 99th mean)	-0.45%			
Power.ForegroundBatteryDrain.30SecondsAvg (50th)	-0.44%			
Power.DailyForegroundBatteryDrain.Exclusive (50th)	-0.60%			

Additional information

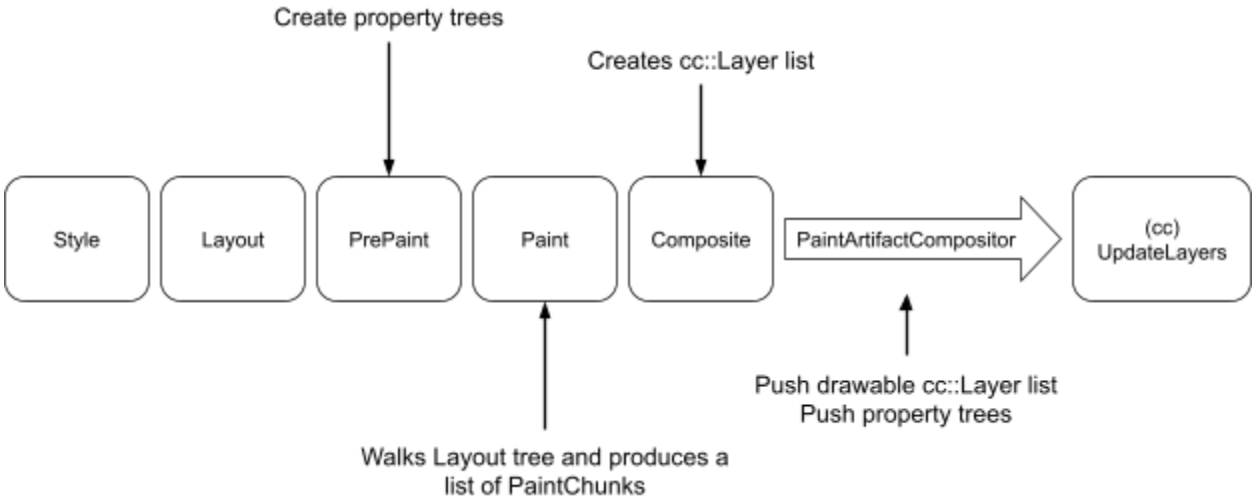
Related documents:

- Perfy award writeup (google only)
- Impact study (google only)
- BlinkGenPropertyTrees one-pager
- PaintNonFastScrollableRegions one-pager
- Simming paint summary
- CompositeAfterPaint testing plan
- core paint/README.md and platform paint/README.md
- Summary of Composite-After-Paint architecture (Part of a BlinkOn 9 talk, April, 2018)

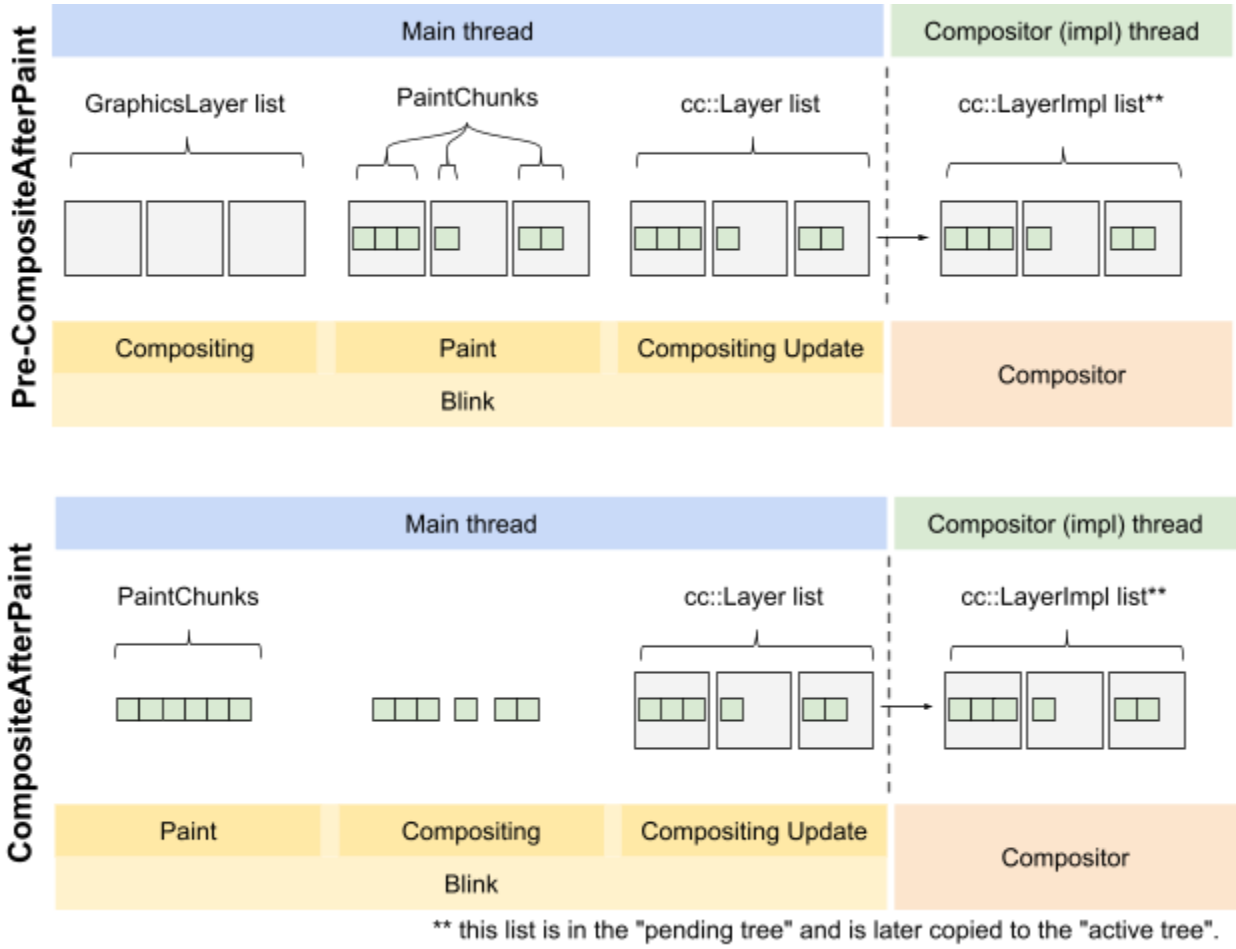
Project tracking sheet
Perf test regression tracker
Pre-CompositeAfterPaint code cleanup tracker

New lifecycle steps

The major main-thread lifecycle steps with CompositeAfterPaint:



Lifecycle changes



** this list is in the "pending tree" and is later copied to the "active tree".