Slimming Paint

Stephen Chenney and Ian Vollick

tl;dr The compositor should consume recordings, not layers.

Project Goals

The Slimming Paint project is a re-write of the Blink and Chromium rendering pipeline. The projects expects to achieve three main benefits:

- Enable more flexibility in compositing decisions without introducing web-visible errors, as our current system would do.
- Achieve performance gains in Blink recording and rasterization that our current architecture precludes.
- Improve the hackability of the rendering code, and hence stability.

Implicitly, we believe that the current system is limiting our ability to be as fast as we can, both in browser performance and software development, and that small incremental changes will not get us to where we want to be.

We are envisioning major changes to the way that we record and integrate with the compositor. At a high level, the idea is that we'll teach the compositor to consume the output from Blink's render tree directly, sidestepping and simplifying a large portion of our current pipeline.

Specifically, the Blink code that manages layers (both Render- and Graphics-) will be entirely replaced, in part by a data container (the PaintList), and in part by compositor code that processes the container to assign content to layers for compositing.

Why? Since the rationale for such a big project is so important, we've created a separate document to hash it out. The high level argument is that this refactor will fix a fundamental correctness issue, improve raster, record and compositing update performance and put us in a good place, architecturally, for the future.

Some prior documents considering much the same issues include Blue Sky Compositing, Composited Recordings, and Rethinking Blink Painting.

Mailing List

silk-dev@chromium.org

Requirements

Details on requirements are in the Composited Recordings Requirements document. The most important requirements are:

- Do not break the web

- Do not go slower
- Do not make the code more incomprehensible

Prototype Workflow

We're living dangerously and working in trunk. Note the known issues document.

Project Breakdown

Gather Data and Design

There are some things we just don't know that would be good to know. For example: what is a typical sequence of layout and paint passes (informs caching of painting data); how much stuff typically intersects a repaint rect (informs the expected gains from partial repaint, plus granularity questions); how do we access style data during painting (informs style data memory changes); etc. With this data, and the requirements, we are in a position to decide what to prototype.

PaintList construction

The aim is to create the artifact that other systems can work with. A PaintList is a list of drawing commands and hints that contains everything you need to produce a final rendering.

Blink, meet Viewport

Make Blink viewport aware, which is a prerequisite to Blink reducing the number of nodes we touch during paint.

PaintList displaces RenderLayer::paint

Drive the generation of SkPicture objects through the PaintList. Instead of calling paint on the RenderTree, we will invoke it through a PaintList manager.

PaintList displaces RenderLayers

Replace the RenderLayer code, and much of the composited layer code, with a new system that processes a PaintList and generates the Graphics layers. This is a partial step to moving all backing decisions to cc.

Compositing makes use of the PaintList

Implement the cc code to consume the Blueprint directly for decisions on backing textures.

Paint only what you need

Take advantage of viewport data, plus layout determined changes, to only update PaintList content that we must.

Shift the Semantic Boundary

Currently, our semantic boundary is the GraphicsContext API layer. Only concepts directly accessible through the GraphicsContext API can be communicated through to Skia and the rasterizing pipeline. We want to move that semantic boundary to the PaintList construction layer

and pass more semantic information through to Skia. The primary advantage is that Skia will have more context when making decisions on how to draw, even different decisions based on the back-end rasterizer. It will also allow smaller mutations to the PaintList object, regardless of how those mutations are eventually passed to Skia (mutate Skia structure or recreate Skia structure).

Related Projects

These projects may unlock additional performance gains when PaintLists are available

Stop chasing Style data

Refactor the way we store style data to reduce pointer walks during both layout and paint. We know we look up style a lot, and we know there is a lot of pointer walking for any given style lookup. There are undoubtedly gains available if we can change this (less work overall and more cache efficiency).

Skia Text Blobs

Text is a longstanding paint bottleneck. The Skia text lobs project aims to develop objects representing chunks of text to draw, with all properties defined as mutable to enable caching and re-use. This is relevant to Slimming Paint because it will provide the caching mechanism for low level text objects, that are otherwise too small for efficient SkPicture caching.

GPU Primitive Batching

Skia's GPU back end performs best when objects that use the same shader are drawn in batches. This reduces context switches in the hardware. It is a well known method from video games. This project informs Slimming Paint of the best candidates for paint order optimization: Reorder text to get common font blocks painting together (color and some other properties can vary but still enable batching); then reorder primitives such as rectangles; then reorder for common paint properties.