# [WIP] Blue Sky Compositing

*The compositor integration of our hopes and dreams*

*Ian Vollick (vollick@)*

### *tl;dr*

*The compositor should consume recordings, not layers.*

## Background

In the early days of chromium's GPU integration, we had not yet blinked and there were other, established compositors to consider. Cooperating necessarily constrained our design, and as a result, our current compositor integration is a bit of a snarl. We've been working hard to unravel it and shoehorn in some much-needed features like layer squashing and universal accelerated overflow scrolling, but it's still kind of a mess. And it's kinda broken. Kinda fundamentally (more on this later). It's also sparsely tested.

### Our World Today

Say we've parsed our HTML/CSS, produced some Nodes, attached some RenderObjects and now we want animate some of those RenderObjects with a CSS3 transform. We need a composited layer. How do we get one? Well first, you need a RenderLayer, and not just any RenderLayer, a *self painting* RenderLayer.
TODO: define terms and explain the paint inversion problem.

## Idea

Composite in terms of draw ops.

- if you squint at them, recordings look a lot like a layer tree
- why not use them *as* the compositing inputs?

## Overview

- Invalidations now stored in blink.
    - Repaint after layout has started this machinery. Just need to store 'new' rect.
- When requested, blink pushes updates to recording into cc.
    - Only updates to renderers in the region-of-interest.
    - Not a full re-recording.
        - ~~Incremental painting updates would be ideal, but this would entail a full rewrite of the painting code in blink. A non-starter for v1.0.~~
            - This is false. We *are* considering a painting rewrite! (Hooray!)
- These recordings must now store compositing info in addition to drawing info. E.g.
    - This batch of commands is affect by an animated CSS3 transform.
    - This batch of commands describes scrolling content.
- Compositor can assign commands to textures based on recorded hints.
    - Geometry problems here are hard. More info in the 'data structure' section.

    - Note: could overlap test here in much the same way gecko does.
- A copy of the recording artifact is maintained on the compositor thread.
    - Ideally a “commit” would involve passing a copy of the recording updates to the compositor thread, not a full copy of the data structure.
    - Must have knobs for realizing our existing compositor-driven effects (for as long as we require compositor-driven effects).
        - Note: I’ve wanted to add code in blink for doing an async updates for a long time now. Rather than calling animateLayers, cc would call a new, thread-safe blink routine that would apply the compositor driven effects for the given frame. This would let all the animation and scrolling code live in blink where it belongs.

## The Magical Recording Data Structure

### Requirements

- Support partial updates.
- Compositor data must be mutable (for compositor-driven effects).
- Must be able to efficiently figure out what render surfaces and textures are needed, what their geometry should be, and which draw ops should contribute to which target.
- Must be able to efficiently determine if any draw op affects a given tile.
- Must be cheap to maintain a copy on the compositor thread.

TODO: propose a data structure and write some pseudocode for key routines.

The exact data structure depends on the compositor integration. For example, if we were to do DOM to texture for WebGL, we’d have a different compositor integration that wouldn’t need to do as much geometry computation (as it would be handled by the WebGL application). For the current, cc/skia integration, this will look a lot like a hierarchical SkPicture. The nodes in this hierarchy will likely not be defined in skia as they will own compositing-specific information that is irrelevant to rasterization. These nodes will aggregate SkPictures. This data structure will also likely need to reuse much of the functionality provided by picture piles to allow for cheap reraster, but this is debatable.

Another unknown is how we’ll make it cheap to get a copy of the data structure to the compositor thread. We could double/triple buffer. We could pass around deltas (assuming that deltas are even feasible to generate; they may not be for SkPictures, eg).

## The Magical Blink Spatial Algorithms

TODO: propose a plan for efficiently figuring out what needs to be recorded. Insert spatial data structure idea here.

## FAQ

### Q: How can we assign textures/surfaces based on just recordings?

- Answer: It's true that armed with just an SkPicture tree, we wouldn't have enough information to make these decisions; we'd have to make our decisions based on previous recordings. The result would be a catastrophic mess of unnecessary rasterization and layer churn as we discover texture candidates. We could never do this. Instead, we have to decorate the recording with enough semantic hints for the compositor to make the right calls. NB: this is a marked change from the current painting code which does not paint "semantics." Our recording will need to be modified.

### Q: But won't that pollute the recording with css/blink-isms?

- Answer: Done poorly, the augmented recording could indeed become polluted, but that need not be the case; it depends on what the hints look like. For example, we might flag a contiguous swath of draw commands as "moving coherently" or "will-change-transform." The compositor would treat this as a strong texturization hint. The reasons for applying the "moves coherently" hint may be based on css/blink-isms (due to scrolling or a CSS3 animation, say), but the hint itself is domain-agnostic and could easily be used by any compositor client.

### Q: What about software raster?

- Even if we get to Ganesh "everywhere" we're still going to have blacklisted drivers and cards and we need to have a reasonable software path. How does a paint data structure fit in this world? Do all layers and caching move behind this recording data structure, and Skia has to handle software vs hardware caching heuristics? Do we maintain a separate path and have all of the cc machinery live on forever?
- Answer: The plan depends only on impl-painting (so we can record draw operations rather than pixels), and in software mode, we would software raster and tex upload as we do currently. Caching would indeed need to move behind the data structure as it would subsume the layer tree.

### Q: What about compositor-driven effects?

- Answer: we would not necessarily lose the compositor thread. While threaded compositing is still valuable, we would need to record enough information for the compositor to drive animations, scrolling and for video to work. This means that the compositor must be able to mutate the properties that drive these effects. It must also be able to read and write from the data structure without contending with the main thread. This should be possible, given the design of the data structure (TODO: link to that section of the document).

## Q: How can you do this without regressing record performance?

- Layers are a sparsity abstraction for recording and raster. They also provide a way to move something around or scroll the screen without having to rerecord things that haven't been invalidated. How can that be achieve without layers?
- Answer: The assumption is that painting will modify a persistent recording data structure, and that the modifications will be minimal. Scrolling and animation should just update a single entry in this data structure. (In fact, in an ideal world, invalidation would be draw-operation rather than spatial region, so we could rerecord just a color, etc.)

## Q: How do we coordinate main-thread and compositor-thread driven effects?

- Answer: We tick in both places. We can be stale on the main thread if we do this, but this is no worse than the current situation.

## Q: How do we reconcile conflicts/staleness?

- Answer: In the case of an identical animation on the main and compositor threads, we'll do the same as we do currently and use the fresh value computed for this frame on the compositor thread. For scrolling, we can either do what we currently do and plumb back scroll deltas to the main thread, or we could instead process scrolls on both threads simultaneously.

## Q: How and when does blink tell cc about dirtiness? How does this work during composited scrolling?

- Answer: We'll record the bounds of the object, even when we don't record the whole thing, so that cc can know that we're about to show missing content and checkerboard as necessary.