Surfaces in Chromium

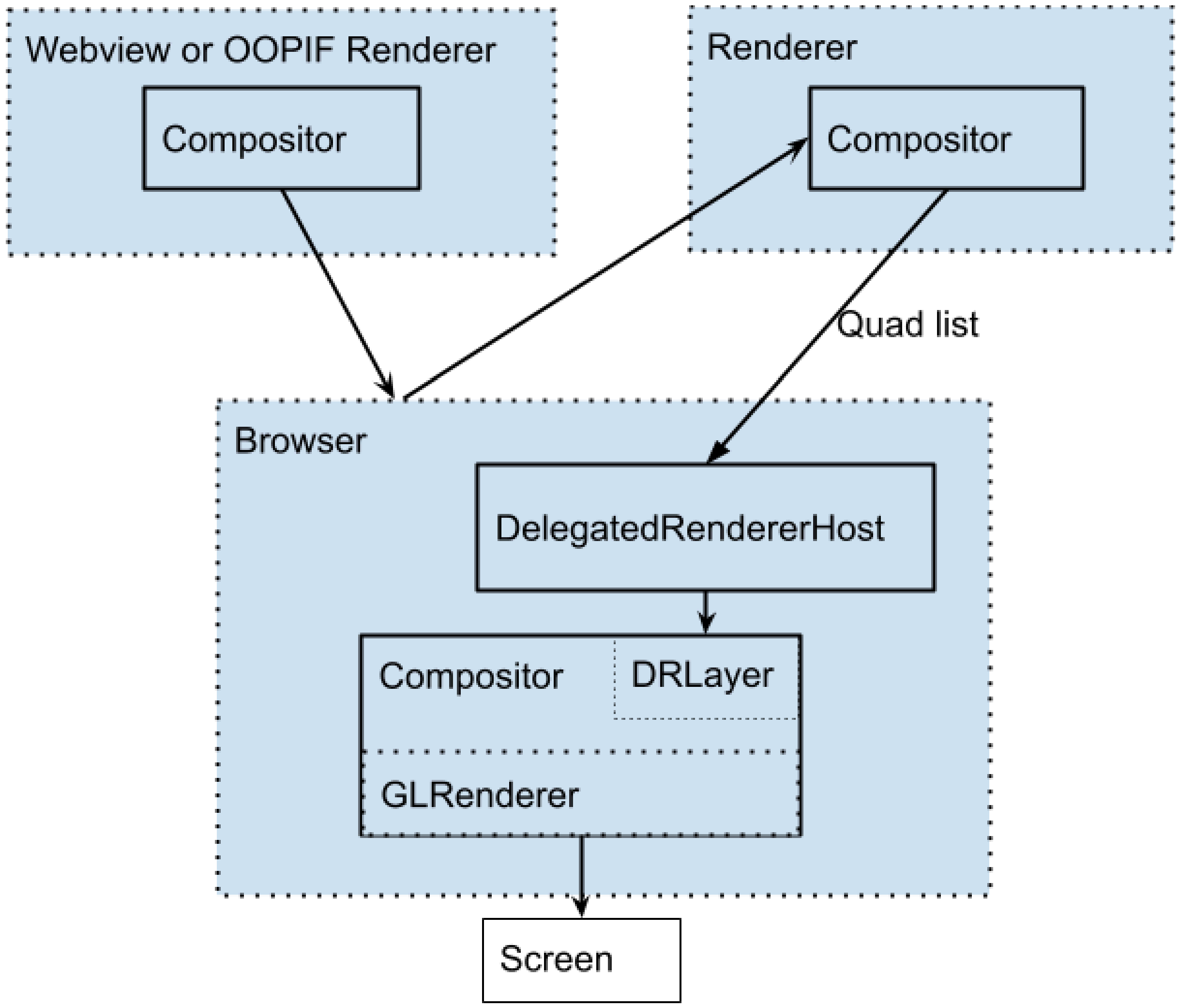
jbauman@chromium.org

Status

Surfaces are used for rendering on every platform. In layout tests the renderer compositor draws everything directly without using delegated rendering or surfaces.

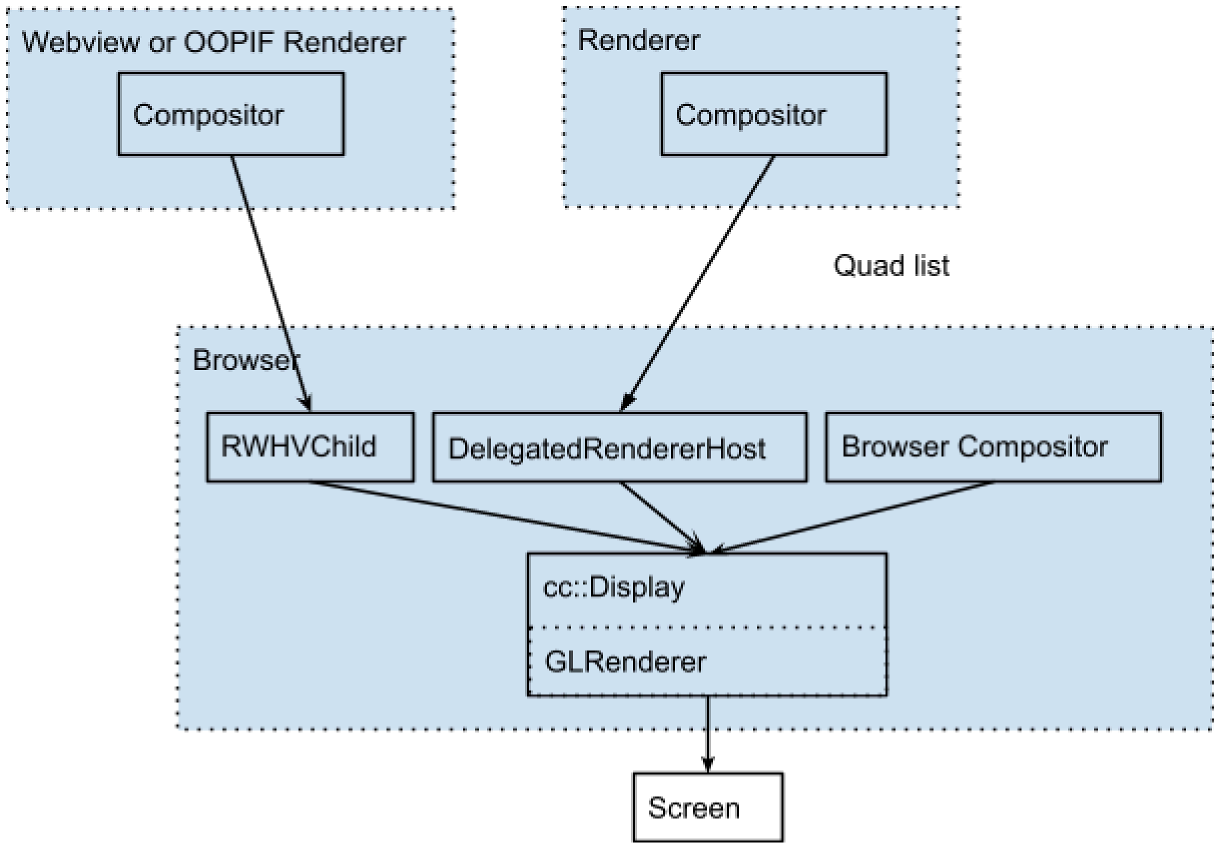
Background

Previously, Chrome used delegated rendering. Every renderer compositor outputs a list of quads that describes how to draw a frame. The browser process takes this *delegated frame data* and associates it with a layer. There's a browser compositor associated with every OS window, and the layers are attached to the browser compositors. Whenever a browser compositor draws, it converts its data to quads and merges in the quads from child renderers, then uses a *GLRenderer* or *SoftwareRenderer* to draw the contents to the screen.



Surfaces (https://docs.google.com/document/d/1RxbffpK_GxPtZscXgIEN0N9ZT7IC8BObnbx9ynw92qg/edit?usp=sharing) are a new mechanism for merging content rendered by multiple compositors together. Every compositor acts like a delegated renderer, and outputs quads. There's a final Display component that merges all the quads together - a SurfaceDrawQuad is used to signal where a quad needs to be included in another.

Surfaces inside Chrome



(Detailed diagram at https://docs.google.com/drawings/d/1-GuIwC2Ta8KSiPuOHvc2aI4rXLUZ03CReiTUTbqeehA/edit?usp=sharing)

Every renderer gets its own Surface that contains its frame data. The browser compositor will use a SurfaceLayer to represent the contents of a Surface in its layer tree.

Every browser compositor will have a cc::Display associated with it - there will be a Display per window that's drawn to. The browser compositor won't have its own GLRenderer, but will output delegated frame data with a quad list into a Surface. The Display will merge it all together and handle the rendering.

Android Webview

Android webview currently has a very simple *parent compositor* that handles transforming and clipping contents from the renderer, but nothing else. With Surfaces this would be replaced with a cc::Display that takes in a Surface that's manually-constructed to handle those transformations and references the Surface from the renderer. The parent compositor itself would be removed, so there's nothing analogous to a browser compositor from normal Chrome.

Advantages over delegated rendering

- Aggregation of multiple surfaces into a Display is simpler and less-cpu intensive than committing and drawing the browser compositor, so renderer-only updates are cheaper.
- Renderer updates don't have to go through the browser compositor, so scheduling is simpler and has lower latency because renderer updates don't have wait for rasterization, tree activation, or other work in the browser compositor
- Frames from webviews or out-of-process iframes don't need to be transmitted from the browser process, into the embedding renderer, and back into the browser. This reduces CPU work and latency. It may also be possible to use the Surface data for hit-testing, reducing latency for processing mouse events.

Disadvantages

- Browser compositor updates need to be go through an extra component, so require more CPU time.
- Both Display and compositor handle aggregating and transforming quads, so some extra code duplication. This can prevented to some extent by removing now-obsolete code (rendering to DirectRenderers or using DelegatedRendererLayers) from the compositor.

Potential future work

- Get rid of the DelegatedRendererHost and let all renderers create Surfaces directly (might still need a global memory manager).
- It's possible to move cc::Display into the GPU process instead of the browser, which could increase efficiency or flexibility.