Scheduling to avoid checkerboard in Chrome

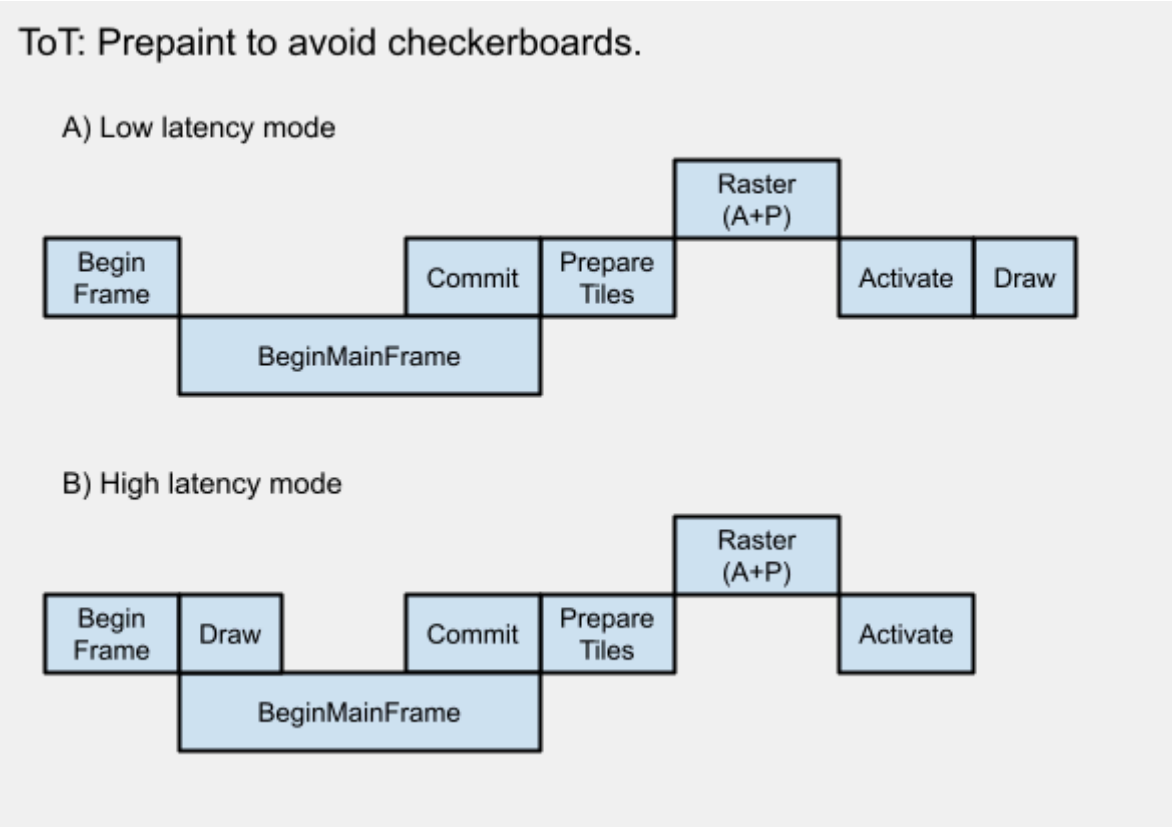
Author: brianderson
Last Modification Date: July 30, 2015
Status: On hold for a potentially better/simpler solution.
WIP Patch: https://codereview.chromium.org/1131633003

Summary

Chrome's compositor currently draws immediately after scroll in high latency mode and relies on pre-rasterization of offscreen content to avoid flashes of checkerboard. That approach has throughput and latency benefits, however it doesn't always work. The solution proposed in this document starts off using the existing approach, but falls back to a more expensive approach to wait for rasterization of visible tiles once we detect checkerboard. How long to wait for rasterization to complete will depend on how the user is interacting with the page.

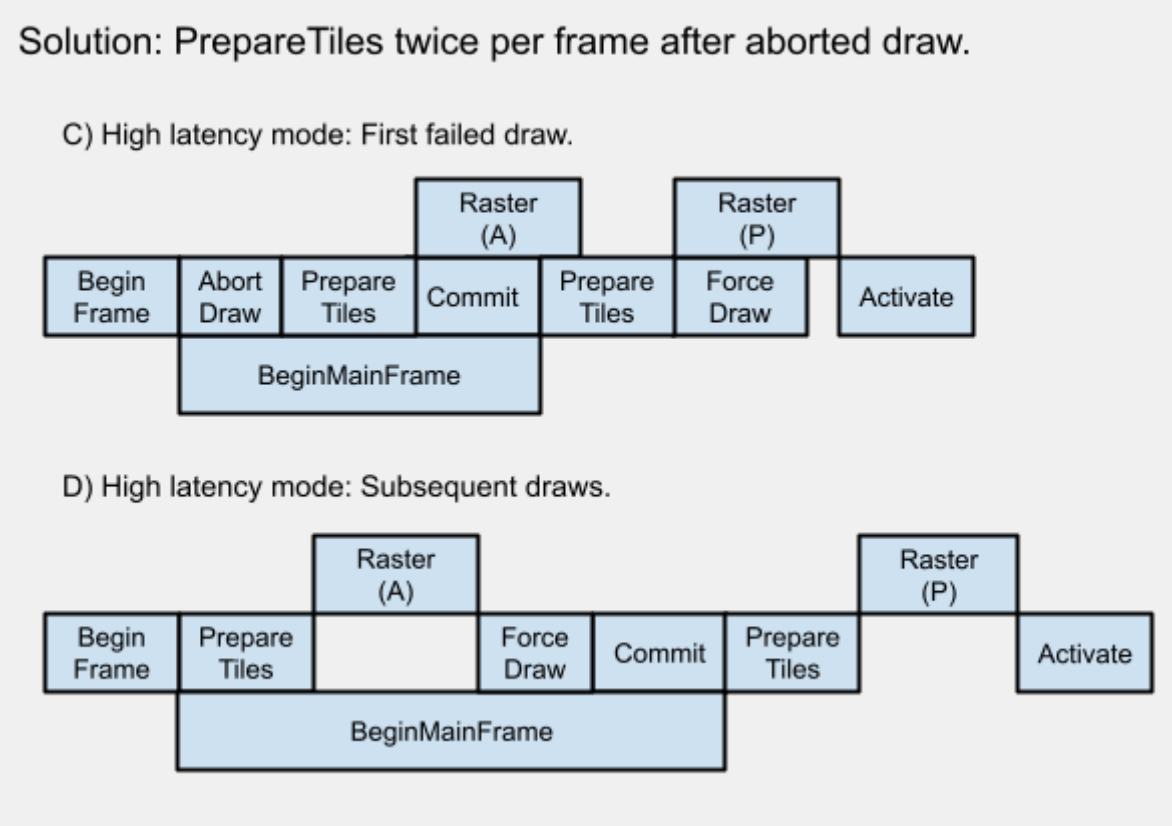
Basic Background and Solution

Below is a diagram of how Chrome currently behaves at ToT when the main thread is in A) low latency mode and B) high latency mode. Both diagrams assume we are hitting 60fps. For slower than 60fps use cases see the "Deadline Selection" part of this doc.



Mode A (main thread low latency mode) will not checkerboard since it waits for rasterization to complete before drawing. Mode B (main thread high latency mode), however, is susceptible to checkerboard since the Draw happens immediately after the BeginFrame without explicitly waiting for visible tiles to rasterize. Scrolls are often associated with the BeginFrame and may have exposed new tiles that haven't been rasterized yet.

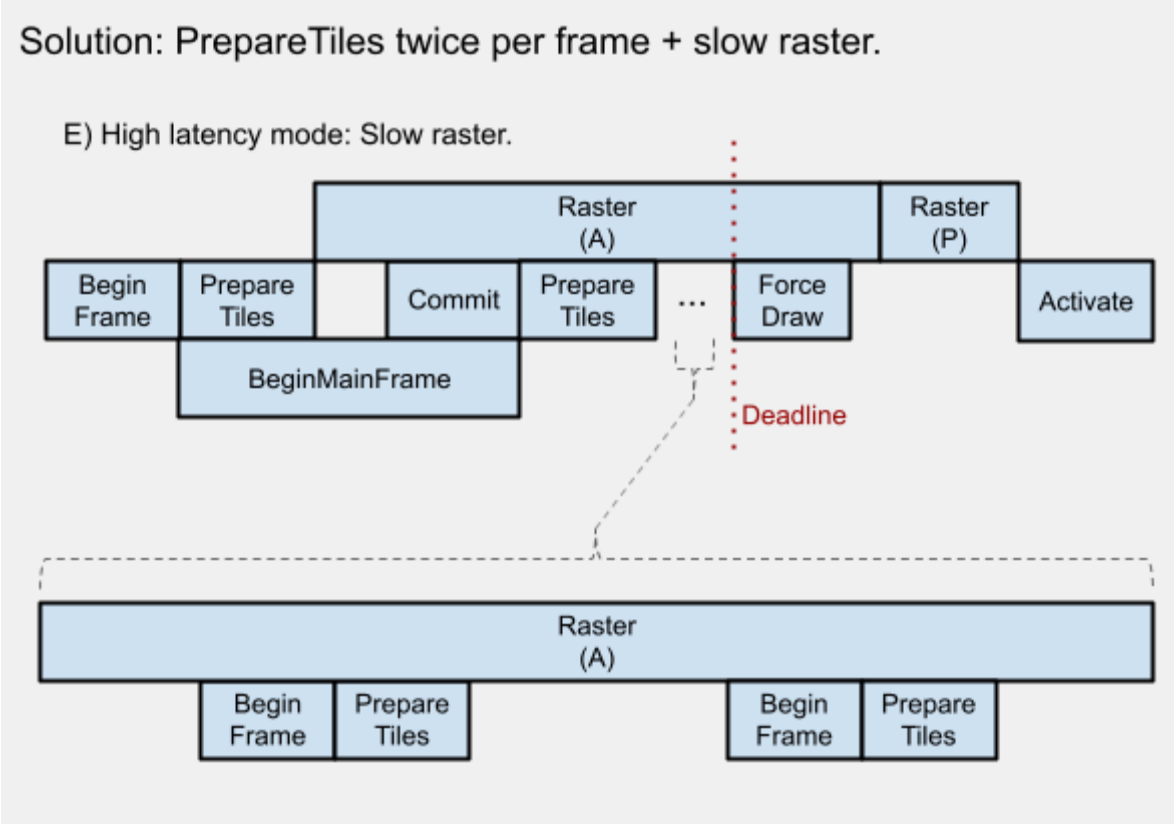
The planned solution is to call PrepareTiles twice per frame, once immediately after BeginFrame and again after commit, as shown in diagram D below. There is a transition to the new mode after an aborted draw as shown in diagram C. Calling PrepareTiles after BeginFrame allows us to get an accurate NotifyReadyToDraw from the TileManager, which corresponds to the end of "Raster (A)" in the diagrams below.



Appendix A provides on overview of potential solutions considered but not taken.
Links to editable diagrams and more detailed diagrams.

Deadline Selection

Rasterization might take a long time and we don't always want to wait forever before we draw. At some point it'll be a better user experience to checkerboard and provide some feedback rather than not draw at all. The diagram below describes what will happen when active tree rasterization takes many frames to complete.



A variable deadline will be selected that depends on how the user is interacting with the page:

- If the user is touch scrolling, the deadline will be ~16ms (60fps).
- If the user is using the mouse wheel to scroll, the deadline will be ~50ms (20fps).
- If the user isn't interacting with the page, the deadline will be ~100ms (10fps).

During this period, additional BeginFrames and scroll events might arrive. Ideally, we would defer both until after we draw to avoid changing the layers in the middle of rasterization, however scroll event deferral is a large effort on its own, and we can get most of the benefit by deferring BeginFrames and associated animation updates.

Implementation Details

Patches will land roughly in the following order:

Minimum changes needed for us to see a benefit:

- Split SchedulerStateMachine::UpdateState into WillAction+DidAction
    - Note: UpdateState currently behaves like WillAction.
    - This will allow us to update state in response to the draw result immediately.
- Control "requires high res to draw" from the scheduler.
    - If it's not controlled by the scheduler, the scheduler's decision to call DrawAndSwapForced could fight with LayerTreeHostImpl's current logic handling "requires high res to draw".

- Make DrawAndSwapIfPossible always abort on any checkerboard.
    - Rename to DrawAndSwapIfHighRes.
- Use DrawAndSwapForced to draw on the deadline after an aborted draw.
- Do not ignore the DrawAndSwapForced return value in cc::Scheduler.
    - It will be used for normal draws now and should update state accordingly.
- After first aborted draw, call PrepareTiles on BeginFrame if needed.
    - Will add a “should_prepare_tiles_on_begin_impl_frame” flag to SchedulerStateMachine.
- Restructure BeginFrame intervals to avoid advancing frames after an aborted draw. There are two options for this:
    - Retry BeginImplFrame or deadline after an aborted draw due to checkerboards.
    - Decouple deadline from frame boundaries.
- Use a deadline that targets 60fps for all cases initially.
    - Will be an immediate improvement without added complexity.

Additional changes that will be implemented

- Treat touch, wheel, and animation checkerboards with different deadlines.
    - Coordinate with InputHandlerProxy to determine which mode we are in.
- Add new DrawResult abort reason for missing RasterSource.
    - If RasterSource is missing, retry BeginImplFrame with a new commit instead of just retrying the BeginImplFrame by itself.
- Remove forced redraw logic.
    - The existing forced redraw logic was intended to address animation checkerboards just in case we are missing RasterSource, which will not longer be necessary after we know with 100% certainty if we are missing RasterSource or not.
- Defer scrolls via BeginFrame.

Additional changes likely not worth the effort

- Differentiate between checkerboard due to queued work vs. unqueued work.
    - If work is already queued, there is no reason to PrepareTiles again.
    - Determining this might be difficult since the layers may have changed since the last PrepareTiles call.
- Initiate PrepareTiles from the scheduler only.
    - This would have helped a lot with the solution in Appendix A since LayerTreeHostImpl unconditionally calls PrepareTiles on commit which is incompatible. The currently proposed solution doesn’t have this problem.

Appendix A: A solution not taken

The diagram below represents a solution where can get away with calling PrepareTiles once per frame (immediately after BeginFrame) and still get an accurate NotifyReadyToDraw from the TileManager. The drawbacks to this approach are 1) the pending tree tiles don’t start rasterizing

until the next BeginFrame and 2) there is an extra frame of latency since Draw, Activate, and Commit happen in the least ideal order. The drawbacks of this approach are not worth the benefits of only having to call PrepareTiles once per frame since we only expect to enter the fallback approach in a small percentage of frames.