Use Zooming to implement Device Scale Factor

Mitsuru Oshima oshima@chromium.org

For background and initial discussion, see the proposal document.

Tracking bug: crbug.com/485650

- Goal
- Design
  - Compositor:
  - WebViewImpl:
  - Coordinates:
  - Scrollbar:
  - Other elements that needs specific coordinate conversions:
    - Popup (select, date picker)
    - DevTools
    - Autofill
    - IME/other elements that depends on focus/selection
    - Pepper plugins/Flash
- Launch Plan:

Goal

- Fix the various HiDPI related issues by using Blink's zoom mechanism.
- high DPI aware APIs should work as before/intended.
  - devicePixelRatio (tested)
  - HighDPI canvas (tested)
  - media queries (tested)
  - css imgset (tested)
  - srcset (tested)
  - picture (tested)
- Migrate all platforms to the new scheme.
  OSX/iOS's device scale factor is always integer, so this is less important on these platform. Nonetheless, it'd be better to have uniform, consistent rendering path for all platforms.

Advantages (In addition to fixing existing issues)

- This allow us to eliminate existing device-scale-factor code from Blink (almost).
- Better font rendering with subpixel font rendering for semi HD displays.

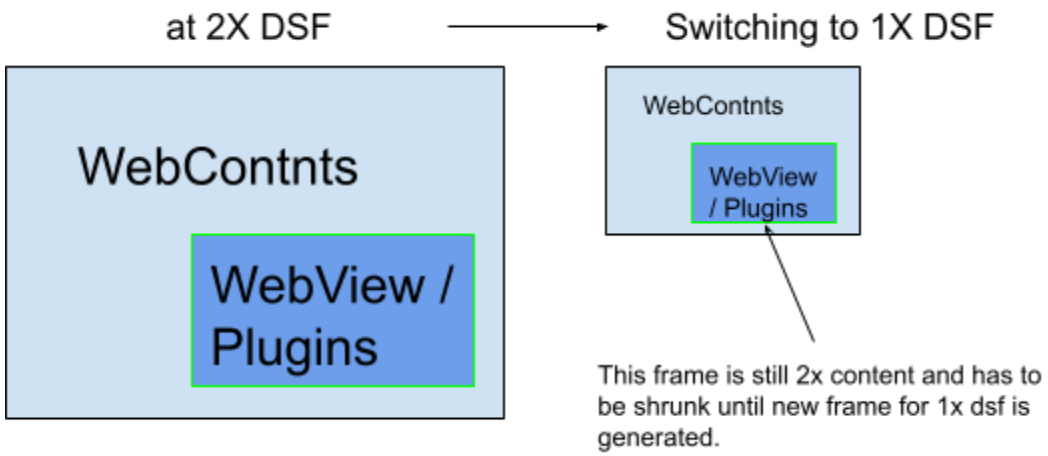
Design

Overview:

Blink always layouts and paints contents with DSF=1, but applies zoom factor that is equivalent to the device scale factor (125% for 1.25dsf, 200% for 2dsf, and so forth). Blink should also uses *Blink viewport* coordinates (viewport origin + css pixel * zoom factor with dsf=1) when communicates with browser/renderer (It currently uses multiple coordinates), and the client is responsible for converting it to target coordinates, such as screen coordinates.

Chrome Compositor will have two device-scale-factors: one used to draw the content inside layer, and the new one to tell for which device scale factor the layer's content was generated for. Additional scale factor (the latter one) is necessary to handle the scale factor transition smoothly because the host renderer (browser compositor) has to aggregate multiple surfaces that might have been generated for different scales. Without this information, a user may see 2x sized content on 1x display (or vice versa). This new scale factor is named painted_device_scale_factor to avoid the confusion.

Exiting (former) device scale factor can be eliminated but requires more work and discussion. (see this doc for details). For now, cc/ will have two scale factors.



~~Option 1:~~

- ~~device_scale_factor: used to scale the content inside renderer.~~
- ~~display_scale_factor/target_scale_factor: used to tag and scale the frame to match the host renderer's scale.~~

  ~~Pros: developers are familiar with what"device_scale_factor" does.~~

~~Cons: It's no longer the scale factor for the device.~~

~~Option 2:~~

- ~~content_scale_factor: this replaces existing "device_scale_factor" and used to scale the content inside renderer.~~
- ~~device_scale_factor: used to tag and scale the frame to match the host renderer's scale. This will also be used in a few place that depends on the device_scale_factor, such as adjusting scrollbar thickness, scaling the input from DIP to Blink's viewport coordinates.~~

  ~~Pros: "device_scale_factor" matches the device's scale factor.~~
  ~~Cons: require more changes (although it's just renaming).~~

~~I have slight preference for option 2 although not strong.~~

Compositor:

- For device-scale-factor=X, renderer renders the content using X00% zoom with device-scale-factor=1.
- Browser UI's compositor uses the single device scale factor for both.
- Renderer's compositor sends frame data tagged with with the scale factor to be painted. That is, when the desktop's dsf=2, renderer's compositor renders contents with dsf=1, but sends the frame with dsf=2 so that the UI compositor can composite as dsf. This is necessary to handle the device scale factor switch smoothly.

```
class LayerTreeImpl {
  void SetPaintedDeviceScaleFactor(float scale_factor);
  // For browser's compositor, this is equal to device_scale_factor.
  float painted_device_scale_factor_;
}

CompositorFrameMetadata LayerTreeHostImpl::MakeCompositorFrameMetadata() const {
   metadata.device_scale_factor = painted_device_scale_factor_;
   …..
}
```

- The guest view's compositor also renders the content with 200% zoom. The frame generated by a guest view will be composited in the context of renderer's compositor which as DSF=1, so it has to be canceled to match the real device scale factor. This can be done in either in BrowserPlugin or ChildFrameCompositingHelper who knows the renderer's context.

```
void BrowserPlugin::OnSetChildFrameSurface(..., float scale_factor, ...) {
  float viewportTo = target-device-scale-factor / renderer's content scale factor;
  compositing_helper_->OnSetSurface(..., scale_factor / content_to_device_ratio, …);
}
```

WebViewImpl:

WebViewImpl will keep a separate value for device scale factor, which will be applied to the frame's zoom factor. The device scale factor will not be applied to the page's scale factor.

```
class WebViewImpl : public WebView  …. {
   private:
     float m_deviceScaleFactor;
}

double WebViewImpl::setZoomLevel(double zoomLevel) {
   …

   if (!WebLocalFrameImpl::pluginContainerFromFrame(frame)) {
      float zoomFactor = m_zoomFactorOverride ? m_zoomFactorOverride :
static_cast<float>(zoomLevelToZoomFactor(m_zoomLevel));
      frame->setPageZoomFactor(zoomFactor * m_deviceScaleFactor);
   }
   ….
}

void WebViewImpl::setZoomFactorForDeviceScaleFactor(float scaleFactor) {
   ….
  m_zoomFactorForDeviceScaleFactor = scaleFactor;
  setZoomLevel(m_zoomLevel);
}
```

Coordinates:

- Coordinates used in Browser/RendererHost will stay in DIP (with a few exceptions)
- Visual Viewport in Blink will become native pixel rather than DIP. Blink Coordinate spaces page will be updated once all platforms are migrated.
- blink::WebInputEvent's coordinate will become VisualViewport coordinates. (It's currently in DIP). In short term, the coordinate will be converted in InputRouterImpl, just before events are sent to renderer (CL).
  In longer term, we'll migrate all WebInputEvent use in browser process to ui::Event, as they expect the coordinates in DIP. The coordinates will be converted when WebInputEvent is constructed. The design doc is here, and it's tracked in

crbug.com/563730.

Other approaches considered:
1: Convert the coordinates in renderer: In InputEventFilter
Since the event is decoded in InputEventFilter and passed to several type of handlers from there, InputEventFilter seems to be the best place to convert the coordinate in WebInputEvent. Unfortunately, InputEventFilter currently have no good way to access device scale factor of the target. One approach is to pass the device scale factor along with WebInputEvent and convert it in InputEventFilter. Here is a trial CL to demonstrate this approach. This looks overkill compared to the proposed approach.

2): Get the device scale factor from RenderViewImpl in InputEventFilter
This seems to be quite challenging given that it has no context of the receiver yet.

- RenderView and WebWidgetClient will have a method to convert VisualViweport to Window, which will be used to convert the coordinates by Blink client and Blink itself respectively.
  RenderWidget/RenderViewImpl will provide the implementation for these interfaces.

```
class RenderView {
  public:
    virtual void viewportToScreen(WebRect* rect) = 0;
}

class WebWidgetClient {
  public:
    virtual void viewportToWindow(WebRect* rect) {}
}
```

- To avoid confusion and bugs, we probably should reduce the use of screen coordinates in Blink.
  a. eliminae Element::screenRect(). This seems to be used only in tests.
  b. Popup uses screen coordinates to compute and adjust the size/location of popups. It's probably better to let browser process handle these computation.
  c. AXLayoutObject::computeElementRect needs investigation.
  d. ValidationMessageClientImpl can use viewport + window rect.
  e. EventHandler/InspectorInputAgent uses viewportToScreen to compute the point in screen for event. These seem to be essential, so we'll need a function to convert a point to screen.

Scrollbar:

Zooming does not change the width of the scrollbar, so its size needs to be adjusted in this mode. There are two approaches.

1. Only enlarge the size for high DPI.
2. Enlarge the scrollbar size when zoomed as well.

Non native scrollbars are already enlarged when zoomed, so we may just want to always enlarge the size.

The size of the scrollbar is controlled in FrameView/PaintLayerScrollableArea.

Other elements that needs specific coordinate conversions:

Popup (select, date picker)

~~Here is the current code:~~

- ~~The size and position of the popup (select tag, date picker) gets converted to screen coordinates in PopupMenuImpl.~~
- ~~Then passed to listPicker.js/pickerCommon.js~~
- ~~Which adjust the bounds using viewport coordinates, and then passes the result to WebWidgetClient (RenderWidget)~~

~~Popup always operates at 100% zoom and uses CSS pixel (= DIP in this context) to control its size, and screen coordinates in DIP to compute the window bounds. However, date picker does not have the same mechanism and does not get scaled with zoom. This will be changed/fixsed in crbug.com/555326 and zooming will be used to scale the content instead. I'll update the doc once it is fixed.~~ This is now fixed. The additional scaling will be applied to the content and anchorRect will be converted using the conversion method explained earlier.

DevTools

DevTools JS code sends the inspected page's rect in viewport coordinates to the DevToolsUIBindings/DevToolsWindow, which is in chrome/browser/devtools.
I think this is simple enough that we can just convert it on the chrome browser side instead of changing the dev tools impl.

The device emulator code needs to be updated to use zooming parameter instead.

Autofill

bounding_box will be converted from viewport to window coordinate in ShowXxxPopup methos in components/autofill/content/renderer (eg. PasswordAutofillAgent::ShowSuggestionPopup)

IME/other elements that depends on focus/selection

RenderWidget::GetSelectionBounds method converts the focus/selection rects from viewport to screen.

Pepper plugins/Flash

Pepper plugin API has a notion of device scale factor, and an application can implement its own scaling logic based on that parameter. We probably need to use the viewport to window/DIP in PPAPI.

Tab Capture/Fullscreen in a tab

We need to test if these features works in the same way as before.

Caveat: There may be others that I may be missing. I'll add them as I find.

Launch Plan:

- The new mode will be implemented behind a flag.
- Aura first, android next, then OSX/iOS.
- When all features are implemented for aura, the flag will be turned on by default on CrOS first (M49). Other platforms will follow based on the feedback on CrOS.
- We'll review the known issue and decide if we should keep the flag on or turn it off before m49 branch cut.
- Other platforms will be worked on in the following milestones.
- Once all platforms are migrated, the device-scale-factor code will be removed from Blink core/web because it's always 1.0
    - Updating device scale factor will be applied directly to the RenderWidgetCompositor, instead of WebLayerTreeView.
    - Platform still knows the device scale factor to coordinate conversion.