GPU Rendering Benchmarks (aka Smoothness benchmarks)

Contact: nduca, ernstm

Chrome now has an awesome rendering benchmark system for GPU and rendering related benchmarks. **It works on all chrome flavors, even android and CrOS, even in their content_shell forms**. To run it you need:

- A chrome build. Just canary or a stable will work. Or download a continuous build from http://commondatastorage.googleapis.com/chromium-browser-continuous/index.html
- python.

Once you've got these things, you're ready to go. To run our top 25 page list through our smoothness benchmark (which tests scrolling speed for sites that scroll, or interaction speed for sites that have interactions):

```
mkdir ~/perf # or wherever you want to put the benchmarks
curl -O http://src.chromium.org/chrome/trunk/src/tools/perf/run_measurement
chmod +x ./run_measurement
./run_measurement --browser=canary smoothness tools/perf/page_sets/top_25.json
```

If you've got a chrome checkout of your own (Get the Code), then just do this:

```
tools/perf/run_measurement --browser=canary smoothness tools/perf/page_sets/top_25.json
```

To run the smoothness test on a Chrome OS device with IP address `$CHROMEBOOK_IP` from a host machine with a chromium checkout, do this:

```
tools/perf/run_measurement --browser=cros-chrome --remote=$CHROMEBOOK_IP
smoothness tools/perf/page_sets/top_25.json --allow-live-sites
```

To benchmark impl-side painting on important mobile sites:

```
tools/perf/run_measurement --browser=canary smoothness tools/perf/page_sets/key_mobile_sites.json --
extra-browser-args="--force-compositing-mode --enable-impl-side-painting --enable-deferred-image-decode -
-enable-threaded-compositing"
```

Lets break down this command a bit:

- **`tools/perf`** is where we keep our GPU benchmarks. It contains benchmarks, which are written in Python.
- **`run_measurement`** is the script we use to run a benchmark across a list of pages

- **`--browser=canary`** tells the script to use Chrome Canary, if it is installed on the system. If you dont have canary [eg you're on linux] it'll fail and tell you to give it another browser.
  - `--browser=list` - for all browsers that the script thinks it can use. Pass --browser-list -vvv if you're not seeing a browser you expect to see.
  - `--browser=system` - the stable chrome install on your system
  - `--browser=debug` or `release` - chromium from out/Debug etc, if it was found
  - `--browser=content-shell-debug` - a content shell build found in out/Debug
  - `--browser=android-chrome` - chrome detected on an attached android device via adb
  - `--browser=cros-chrome --remote=$CHROMEBOOK_IP` - chrome running on your chromebook
  - `--browser=exact --browser-executable=<path to build>` - your tests will work with any chrome build >= M18!

- **`smoothness`** is the name of the benchmark to run. If you type `./run_measurement`, you'll see a list of other benchmarks that we support. There are a lot, from **JSGameBench**, to **Dromao**. **Smoothness** is our catch all test for graphics.

- **`tools/perf/page_sets/top_25.json`** is a list of 25 pages that we monitor continuously on our bots. The benchmark you pick will run on these pages. There are other sets of pages too, for example `key_desktop_sites`, `key_mobile_sites`, and `tough_scrolling_cases`. Some have hundreds or thousands of sites. Some have only a few. Pick the one that fits your goal.

When you run this, you'll get some CSV output that looks like this:

```
url,average commit time (ms),average image gathering time (ms),average latency
(ms),average num layers drawn (),average num missing tiles (),average tile analysis time
(ms),average touch acked latency (ms),average touch ui latency (ms),dom content loaded time
(seconds),dropped percent (%),first paint (ms),load time (seconds),mean frame time
(ms),percent impl scrolled (%),solid color tiles analyzed (count),texture upload count
(count),total deferred image decode count (count),total deferred image decoding time
(seconds),total_image_cache_hit_count (count),total_texture_upload_time (seconds),total_tiles_analyzed
(count)
https://mail.google.com/mail/,0,0,41.6191934524,0.0,0.0,0,0,0,0.11,0.0,103.8,1.596,15.162,0.0,0,0,0,0,0,0,0
```

Ugh. Not human readable, but great for a spreadsheet. But, lets try `--output-format=terminal-block`

```
url: https://mail.google.com/mail/
average commit time (ms): 0
average image gathering time (ms): 0
average latency (ms): 22.3321869436
average num layers drawn (): 0.0
average num missing tiles (): 0.0
average tile analysis time (ms): 0
average touch acked latency (ms): 0
average touch ui latency (ms): 0
dom content loaded time (seconds): 0.124
dropped percent (%): 0.0
first paint (ms): 124.2
load time (seconds): 1.619
mean frame time (ms): 13.212
percent impl scrolled (%): 0.0
solid color tiles analyzed (count): 0
texture upload count (count): 0
total deferred image decode count (count): 0
total_deferred_image_decoding_time (seconds): 0
```

```
total image cache hit count (count): 0
total texture upload time (seconds): 0
total_tiles_analyzed (count): 0
```

Now that's useful (once you figure out what the data shows!). These are some key statistics for that page as it scrolled, in the default mode for that platform. But, lets say you wanted to run chrome in one of its super fancy experimental modes, like forced compositing, impl-side painting, the thread and deferred image decode all at once, `--extra-browser-args` is your friend:

```
tools/perf/run_measurement --browser=canary smoothness tools/perf/page_sets/top_25.json --output-format=terminal-block --extra-browser-args="--force-compositing-mode --enable-impl-side-painting --enable-deferred-image-decode --enable-threaded-compositing"
```

Fun! Remember, unless you pass `--disable-gpu-vsync`, scrolling goes only as fast as your screen. So, for screen with 60 Hz refresh, 16.6 is usually a good thing.

Smoothness Metrics

Painting vs Rasterize: throughout the metrics, you will see the words paint and raster. These have very precise meanings:

- paint: time dumping WebKit's rendering structures into the compositor's rendering structures.
  - Software mode, and regular compositing modes: this is the time spent to walk the webkit tree AND software-rasterize its 2D ops AND any time required to do image decodes
  - Impl-side painting mode: this is the time to JUST walk webkit tree and dump it into an SkPicture. IOTW, recording time
- rasterize
  - Zero in software mode and regular compositing modes
  - Impl-side painting: this is the time to rasterize SkPictures to tiles. If we had an decode cache miss, will include time servicing the image cache miss.

With that in your mind, the numbers mean:

- average_commit_time (ms)
  Time spent pushing the layer tree from the main thread to the compositor thread. Is zero if software rendering.
- average_num_layers_drawn
  Number of layers in the tree at draw time. Is zero in software mode.
- dropped_percent (%)
  Number of frames that missed vsync. The metric is slightly different in each rendering mode but roughly approximates how janky the page was.
- first_paint (ms)
  How long it took from navigate for the first frame to be put onscreen.
- **mean_frame_time (ms)**
  The frame rate, but reported as an interval. This is probably what you wanted to see all along, 90% of the time.
- percent_impl_scrolled
  The percent of input events that caused fast scrolling on the impl thread. If you see numbers between 0 and 100, its probably because the page changed halfway through and became slow scrolling, or vice versa.
- texture_upload_count
  The number of textures uploaded to the GPU.
- total_texture_upload_time
  The time spent in texture upload on the GPU process
- jank_count (Android only)
- "Jankiness" is a measure of how smooth an animation appears; the lower this number is, the more fluid the animation looks. This metric tracks how many times during the benchmark we failed to produce a frame in time and had to re-display the previous frame. Specifically, it counts the number of times the delay between successive frames increased by a multiple of the vertical sync period (e.g., 1/60 seconds).

How it works

Telemetry performance testing framework

page scrolling is done by telemetry's "scroll" interaction, tools/telemetry/telemetry/scroll_interaction.py. On chrome, it boots the browser with `--enable-gpu-benchmarking-extension`, which exposes a `beginSmoothScrollBy(numPixels, function() { callback })` API to javascript that that simulates scrolling as would be done by the user. We then use it to move a page down.

The `smoothness` benchmark monitors ~15 signals about this interaction, mostly using the renderingStats() API of `content/renderer/gpu/gpu_benchmarking_extension.cc` as well as *Telemetry's Inspector Timeline API*.

Telemetry provides a way to separate out the measurement process from the interaction process from the actual pages being tested. We then maintain a number of important lists of web pages, some synthetic some real, in `tools/perf/page_sets`, grouped by their kind of importance. `top_25`, `key_desktop_sites` and `key_mobile_sites` are likely of particular interest to users.

Telemetry provides a mechanism to very reliably record a web page and then replay it many times in that exact recorded state. We (Chrome team) cannot make our recordings public since the assets the recording are the property of the site owners. However, we have exposed a utility that anyone can use to make their own recordings:

```
tools/perf/record_wpr --browser=system tools/perf/page_sets/top_25.json
```

This will place a file called `top_25.wpr` in `tools/data` that is an archive of the data required to replay those pages back over-and-over again without deviation.

Adding credentials to test live sites that require a logged in user

As part of GPU testing, we often want to measure the performance of a site like Gmail, or Facebook, that sit behind a login. We do not give out logins for these, but if you have your own, you can put a `credentials.json` in `tools/perf/data` or `~/.telemetry-credentials` in the style of `tools/telemetry/examples/credentials_example.json` with the right logins and telemetry will automatically then login to gmail or facebook for you. Patches are welcome to add support for other sites as well.

Comments

You do not have permission to add comments.