Graphics Context Eviction

Background

Today's graphics drivers do not always handle low video memory (vram) situations gracefully. When a computer's vram is running low, stability not only of Chrome, but also of the entire machine, can be impacted.

- Poor performance
  - Extremely slow performance with NVIDIA's drivers on Linux
  - On certain older Windows drivers and Mac drivers, the system becomes very slow (most recent drivers are doing OK though)
  - https://code.google.com/p/chromium/issues/detail?id=139861
  - https://code.google.com/p/chromium/issues/detail?id=145600
  - https://code.google.com/p/chromium/issues/detail?id=95685
- Chrome + Buggy drivers = crash
  - Multiple reports have been received on both Linux and MacOSX, that when users open too many tabs with heavy WebGL contents, Chrome tab crashes (mostly renderer process crashes, but also browser process crashes).
  - Vram usage should be collected at the time of crashes, so a clear picture can be gained of the correlations between mysterious crashes and vram pressure.
- OUT_OF_MEMORY GL errors are not handled gracefully
  - The compositor expects not to encounter OUT_OF_MEMORY errors
  - One example is Linux with AMD drivers, corrupted compositing results can happen. It's easy to reproduce (just run through WebGL conformance tests).
  - https://code.google.com/p/chromium/issues/detail?id=375957

Table 1 shows the vram consumption for a Google Maps tab (through Chrome's task manager data, thanks to ccameron@chromium.org):

	Windows	Mac OSX	Linux
Tab is visible/active	97M	85M	90M
Tab is hidden	84M	75M	80M

Table 1: vram consumption for one Google Maps tab

It's apparent how easily a power user can exhaust vram by running many heavyweight WebGL apps simultaneously.

It's not desirable to simply ignore VRAM usage and assume that the driver will handle high pressure situation well. As WebGL becomes more widely adopted, the vram pressure issue will only become worse.

Current Situation:

WebGL applications are expected to handle the context being lost and restored (see WebGL 1.0 spec: https://www.khronos.org/registry/webgl/specs/latest/1.0/), so a context evictor can be used as a "heavy hammer" to evict WebGL contexts and later restore them.

Right now we evict the oldest context when one of the following conditions is met:

- drawing buffers reach 16M pixels
- 16 WebGL contexts are created

These conditions are set per renderer. In other words, we don't have a Chrome-wide eviction mechanism.

Solution Proposal

In general:

- Implement an eviction strategy for WebGL for chrome globally. Not per renderer.
- Implement something simple. Avoid complicated heuristics. (Maybe get smarter in the future)

Where does the Evictor lives: in the GPU process

Information we need to collect:

1. Per context vram usage
    - Per renderer vram usage is already collected (they are sent to browser process for TaskManager)
2. Context timing: creation time, last update, etc (should be easy to collect)
3. Context ownership and visibility
    - To which tab a context belongs? (from browser)
    - whether a tab is in focus? visible? (from browser)
    - or track whether a canvas is visible or not (from renderer)

To decide if context eviction is necessary, only (1) and (2) are needed, which the GPU process already have. But to decide which context to evict, (3) is needed.

(3) can be collected in one the two following ways:

- Update the GPU side whenever a tab/canvas state changes
    - pro: always ready to make the decision

    - con: lot of IPC
- Only query 3) when context eviction is necessary
    - pro: very likely throughout the lifespan of a Chrome run, no contexts need to evicted, so no wasted work
    - con: could have noticeable delay when we need to evict a context

Under what condition should we start eviction:

1. OUT_OF_MEMORY GL error is encountered
    - This is too late for this context, some damage is already done. In this case, both the least used context and this context should be evicted.
2. A virtual VRAM threshold is reached
    - By default set a virtual limit, say, 512M
    - If the existing TOTAL VRAM on the platform can be queried, this limit can be adapted to that (say, 70% of total vram)
    - Add a way for power users to overwrite the virtual limit (in about:gpu)
        i. Allow a different limit
        ii. Allow no-limit option (with a warning)
3. By doing this, the current 16 contexts per renderer, 16M pixels per renderer limits are lifted
4. Question: if the platform supports, do we want to query the available vram on a regular bases and base our eviction on that?

Factors to Consider:

- Evicting contexts should follow the following order based on tab states:
    - Contexts from background tabs get evicted first
    - Contexts from tabs that are visible but not in focus get evicted next
    - Contexts from tabs that are in focus should be the last to consider
- Evict the context that is least recently updated.
- What to do if only one tab exists, and it creates a lot of contexts and uses a lot of vram, triggering the evictor?
    - bajones@chromium.org mentioned the case of ShaderToy

Context Lost and Restore:

- Differentiate the situations where a context is evicted by Chrome vs where the context is actually lost
    - Raise a warning bar should be avoided if the context is evicted by Chrome
- When to restore:
    - When the related tab becomes in focus again?
    - When the context become visible again?

Testing:

- Expose the function to set the virtual limit to Telemetry, so a vram pressured situation can be emulated and the eviction strategy can be tested
- Things to be tested:
    - Verify under vram pressure, the right context gets evicted
    - Verify that tab becomes visible again, the context is restored
    - Verify there is no endless loop of evictions